FORNAX

# Word2Vec

Krzysztof Kolasiński - January 2017

# Vector representation of words - motivation

**Some warning:**

- most of the material can be found in the internet
- I use mine notation (which is sometimes different that in papers)
- The notation can change during slides - mostly because of variety of problems

# Vector representation of words - motivation

**NLP** - Natural Language Processing

- If we want to process text or document words must be represented as numbers (or vectors).
- We **start with assumption** of no knowledge about semantic and lexical similarities between the words in our corpus.
- This result with sparse vectors in vocabulary (**one-hot encoding**):

| word | vector | id |
|---|---|---|
| good | $=[1,0,0,0,0]$ | 1 |
| bad | $=[0,1,0,0,0]$ | 2 |
| ugly | $=[0,0,1,0,0]$ | 3 |
| nice | $=[0,0,0,1,0]$ | 4 |
| awful | $=[0,0,0,0,1]$ | 5 |

# Vector representation of words - motivation

## Vector representation of words:

| word | vector | id |
|---|---|---|
| good | =[1,0,0,0,0] | 1 |
| bad | =[0,1,0,0,0] | 2 |
| ugly | =[0,0,1,0,0] | 3 |
| nice | =[0,0,0,1,0] | 4 |
| awful | =[0,0,0,0,1] | 5 |

- The words are really independent, **orthogonality** guarantees that we cannot express one word in terms of other: **linear independence**.
- We just need to store ids of each word.

## Low dimensional representation (Vector Space Models) ???

| word | VSM |
|---|---|
| good | [0.4,0.1] |
| bad | [-0.4,0.1] |
| ugly | [0.1,-0.7] |
| nice | [0.3,-0.2] |
| awful | [0.2,-0.9] |

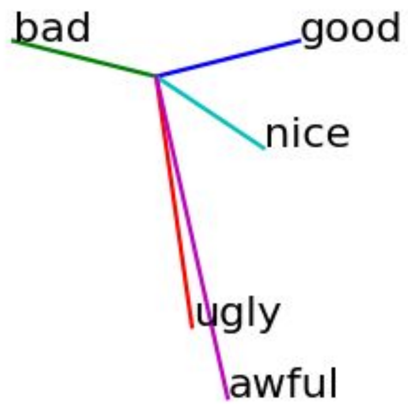
- We want to build a model which will **capture similarities among words**.
- The size of the **latent space is parameter** of the model.
- The model should give that: **similar words are represented by similar vectors** (in terms of some metric - e.g. **cosine distance**)

# Vector representation of words - motivation

## Word embedding (toy example):

| word | vector | id |
|---|---|---|
| good | =[1,0,0,0,0] | 1 |
| bad | =[0,1,0,0,0] | 2 |
| ugly | =[0,0,1,0,0] | 3 |
| nice | =[0,0,0,1,0] | 4 |
| awful | =[0,0,0,0,1] | 5 |

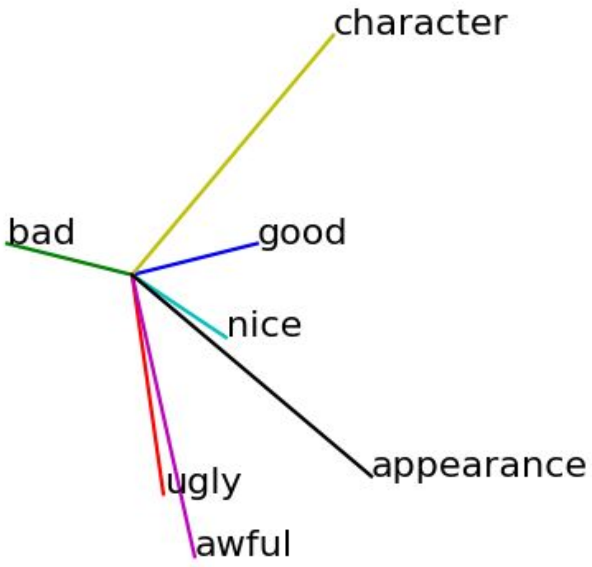
| word | VSM |
|---|---|
| good | [0.4,0.1] |
| bad | [-0.4,0.1] |
| ugly | [0.1,-0.7] |
| nice | [0.3,-0.2] |
| awful | [0.2,-0.9] |




Plotting eigenvectors of covariance matrix of embedded words:

```
words_vec = np.matrix([[0.4,0.1],
                       [-0.4,0.1],
                       [0.1,-0.7],
                       [0.3,-0.2],
                       [0.2,-0.9]])

normalized_words = words/np.linalg.norm(words_vec,axis=1)[:, None]
covariance = wn.T @ wn
eig_vals, eig_vecs = np.linalg.eigh(covariance)
```




**Conclusion**:
size of the latent space determine capacity of the model

# Vector representation of words

## Computing low dimensional representation

| word | vector | id |
|---|---|---|
| good | =[1,0,0,0,0] | 1 |
| bad | =[0,1,0,0,0] | 2 |
| ugly | =[0,0,1,0,0] | 3 |
| nice | =[0,0,0,1,0] | 4 |
| awful | =[0,0,0,0,1] | 5 |

**how to do it?**

**word2vec(x)**

| word | VSM |
|---|---|
| good | [0.4,0.1] |
| bad | [-0.4,0.1] |
| ugly | [0.1,-0.7] |
| nice | [0.3,-0.2] |
| awful | [0.2,-0.9] |

**Distributional Hypothesis** - states that words that appear in the same contexts share semantic meaning.

**There are two groups of approaches for word2vec:**

- ***count-based methods*** - like GloVe (global word co-occurrence statistics)
- ***predictive*** - like CBOW, Skip-Gram (local co-occurrence )

# CBOW - Continuous Bag-of-Word

***1. <u>One word context</u>***

This is nothing else like shallow neural network with one hidden layer:

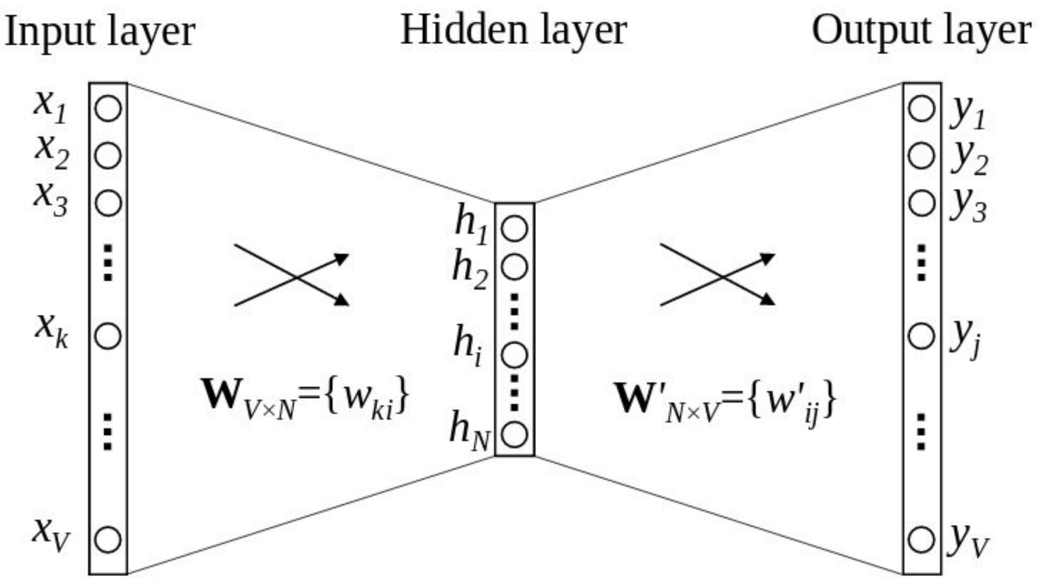
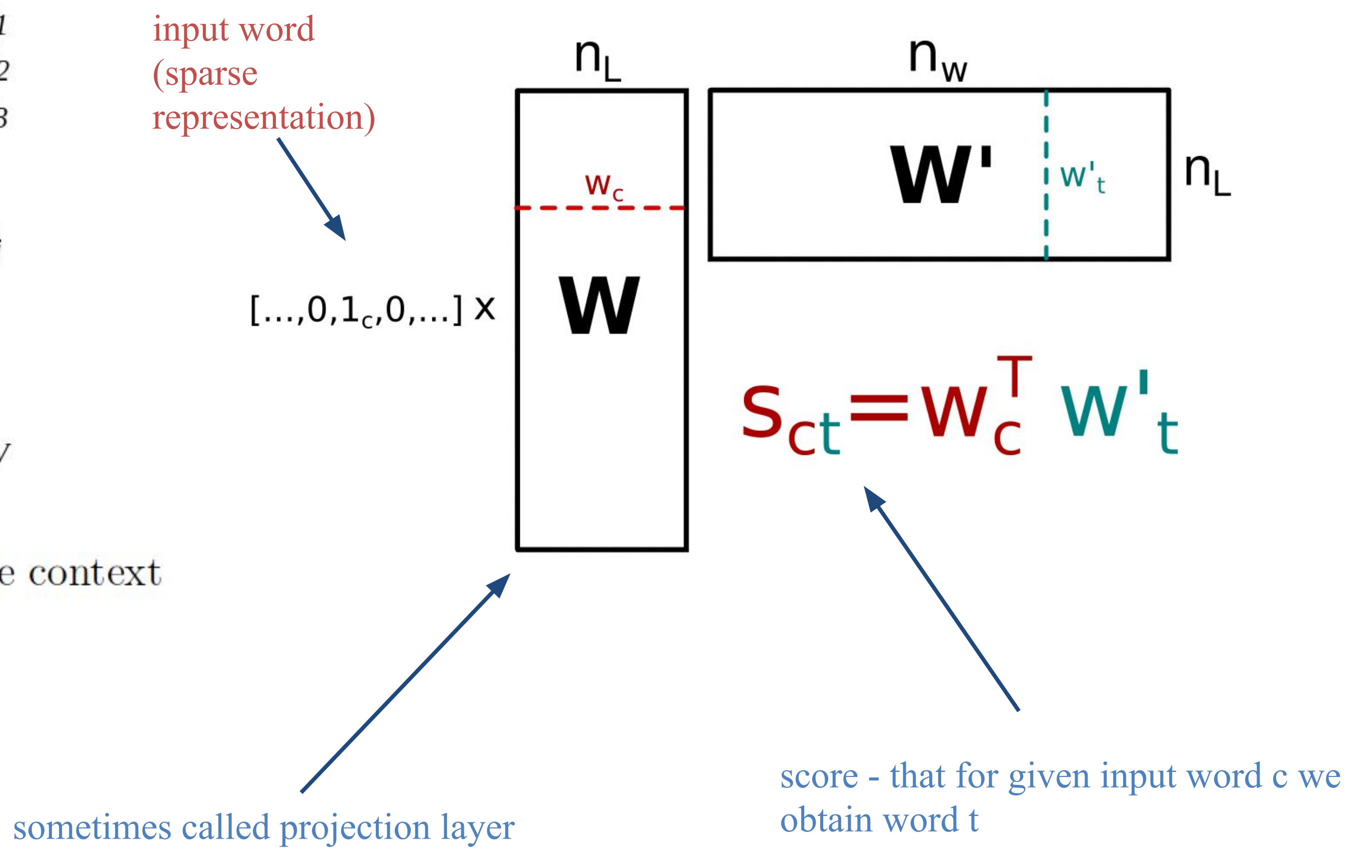



Figure 1: A simple CBOW model with only one word in the context




**<u>Parameters:</u>**

$n_L$ - the size of the latent space

**W, W’** - matrices we are looking for

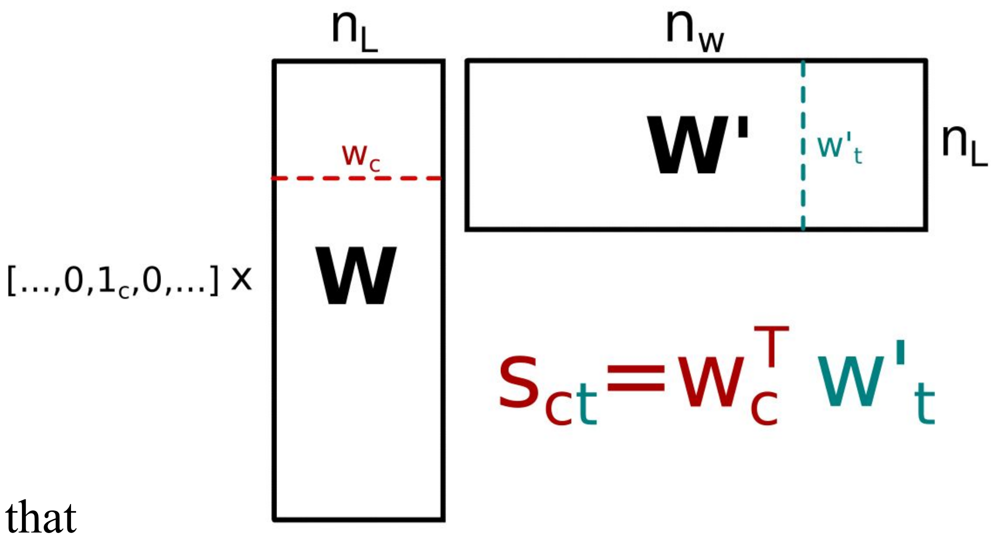
# CBOW - Continuous Bag-of-Word

## 1. *One word context*

This is nothing else like shallow neural network with one hidden layer:

**“the quick brown fox jumped over the lazy dog”**

- We start from creating context/target pairs: (**the** : **quick**, **quick** : **brown**, **brown** : **fox**, ...)
- The words are converted to sparse orthogonal representation: (**1** : **2**, **2** : **3**, **3** : **4**, ...)
- For each input word **k** we will maximize the probability of that the target word (**brown**) will appear after context word (**quick**) i.e. we want to maximize score $\mathbf{s_{kp}}$.



# CBOW - Continuous Bag-of-Word

## 1. *One word context*

### "the math"

For a given context/target pair we compute score

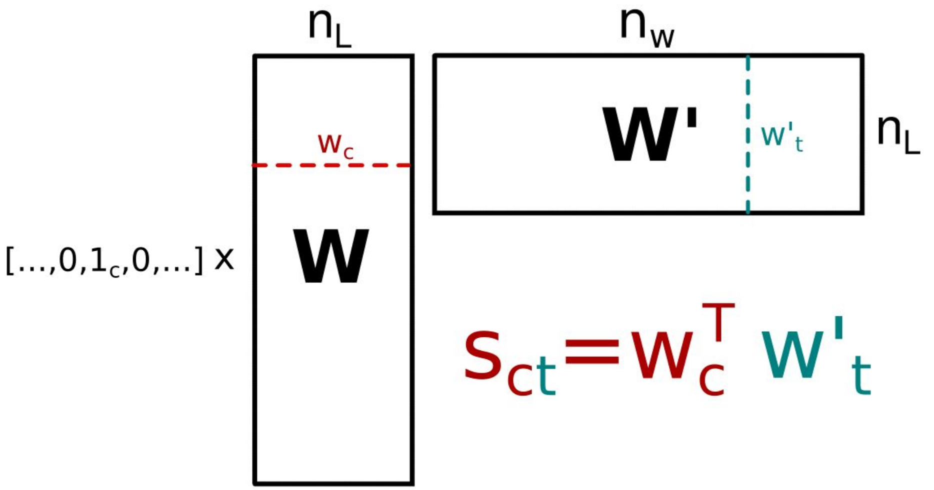
$$s_{ct} = \mathbf{w}_c^T \mathbf{w}'_t$$

The probability of observing word **p** for given input word **k** is computed with **Softmax**

$$p_{ct} = \frac{\exp(s_{ct})}{\sum_i \exp(s_{ci})}$$

The **context/target pair loss** is defined as

$$\begin{aligned} l_{ct} &= -\log(p_{ct}) = -\log\left(\frac{\exp(s_{ct})}{\sum_i \exp(s_{ci})}\right) \\ &= -s_{ct} + \log\left(\sum_i \exp(s_{ci})\right) \end{aligned}$$




The same but expressed in term of **W** and **W'** matrices

$$l_{ct} = -\mathbf{w}_c^T \mathbf{w}'_t + \log\left(\sum_i \exp\left(\mathbf{w}_c^T \mathbf{w}'_i\right)\right)$$

**Total loss** is defined as

$$L = \frac{1}{N} \sum_{\{c,t\}} \left\{ -\mathbf{w}_c^T \mathbf{w}'_t + \log\left(\sum_i \exp\left(\mathbf{w}_c^T \mathbf{w}'_i\right)\right)\right\}$$

**Note!**

The sum represent scan over all words in document: i.e. **word by word**.

**{c,t}** - is the context/target pair

# CBOW - Continuous Bag-of-Word

## 1. *One word context*

### "the math"

For a given context/target pair we compute score

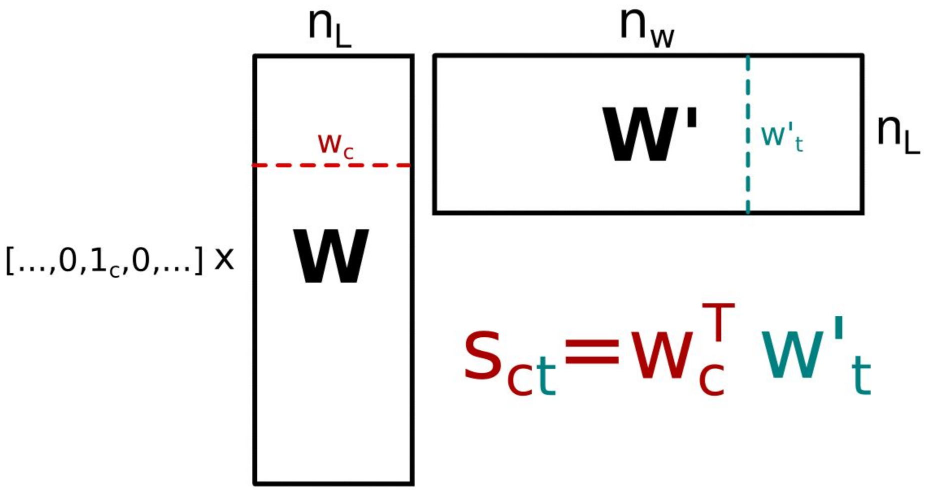
$$s_{ct} = \mathbf{w}_c^T \mathbf{w}'_t$$

The probability of observing word **p** for given input word **k** is computed with **Softmax**

$$p_{ct} = \frac{\exp(s_{ct})}{\sum_i \exp(s_{ci})}$$

The **context/target pair loss** is defined as

$$\begin{aligned} l_{ct} &= -\log(p_{ct}) = -\log\left(\frac{\exp(s_{ct})}{\sum_i \exp(s_{ci})}\right) \\ &= -s_{ct} + \log\left(\sum_i \exp(s_{ci})\right) \end{aligned}$$




Naive implementation of loss:

```
# initialize W and W' matrices
W = np.random.rand(10,2)
Wp = np.random.rand(2,10)
c=2  # context word
t=4  # target word

score_ct = lambda c,t : W[c,:] @ Wp[:,t]
scores_c = [ score_ct(c, i) for i in range(10) ]

prob_ct = exp(score_ct(c, t))/sum(exp(scores_c))

loss_ct = -log(prob_ct)
```

# CBOW - Continuous Bag-of-Word

1. ***One word context***

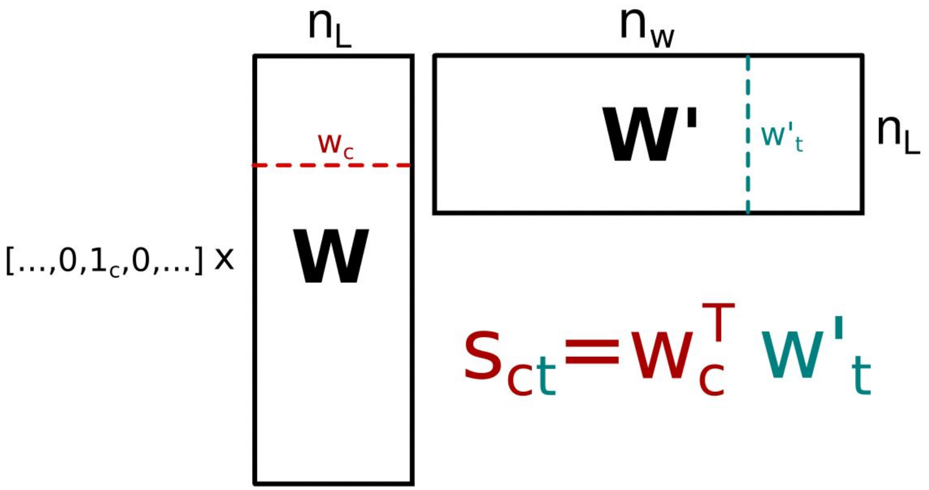
## "the math"

The total loss is never computed

$$L = \frac{1}{N}\sum_{\{c,t\}}\left\{-\mathbf{w}_c^T\mathbf{w}'_t + \log\left(\sum_i \exp\left(\mathbf{w}_c^T\mathbf{w}'_i\right)\right)\right\}$$

Instead we compute it over mini-batches

$$L_B = \frac{1}{N_B}\sum_{\{c,t\}_B}\left\{-\mathbf{w}_c^T\mathbf{w}'_t + \log\left(\sum_i \exp\left(\mathbf{w}_c^T\mathbf{w}'_i\right)\right)\right\}$$




$$s_{ct} = w_c^T\, w'_t$$

The minimization of mini-batch loss is done with Gradient descent:

$$\mathbf{W} := \mathbf{W} - \lambda\frac{\partial L_B}{\partial \mathbf{W}}$$

$$\mathbf{W}' := \mathbf{W}' - \lambda\frac{\partial L_B}{\partial \mathbf{W}'}$$

Or we can look at vector update instead of whole matrix:

$$\mathbf{w}_k := \mathbf{w}_k - \lambda\frac{\partial L_B}{\partial \mathbf{w}_k}$$

$$\mathbf{w}'_k := \mathbf{w}'_k - \lambda\frac{\partial L_B}{\partial \mathbf{w}'_k}$$

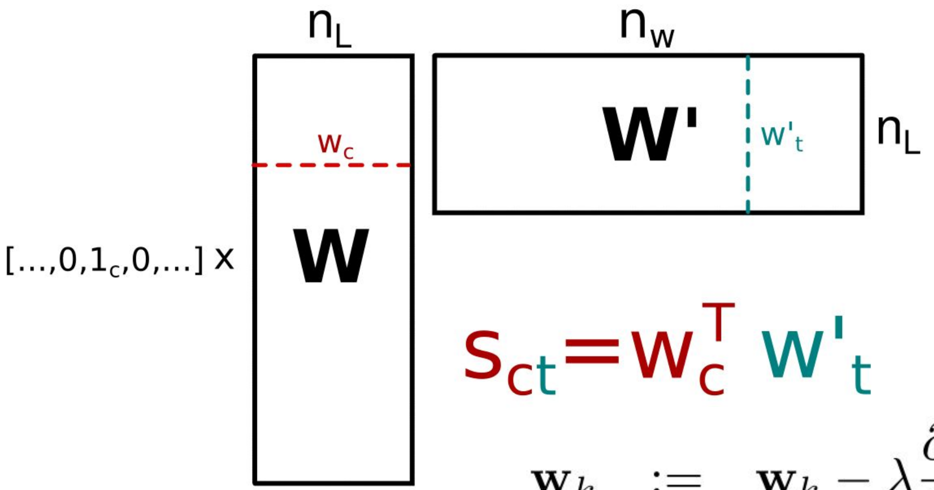
# CBOW - Continuous Bag-of-Word

## *1. One word context*

### "the math"

The total loss

$$
\begin{aligned}
L_B &= \frac{1}{N_B} \sum_{\{c,t\}_B} l_{ct} \\
&= \frac{1}{N_B} \sum_{\{c,t\}_B} \left\{ -\mathbf{w}_c^T \mathbf{w'}_t + \log\left( \sum_i \exp\left( \mathbf{w}_c^T \mathbf{w}'_i \right) \right) \right\}
\end{aligned}
$$




$$s_{ct} = w_c^T \, w'_t$$

$$
\begin{aligned}
\mathbf{w}_k &:= \mathbf{w}_k - \lambda \frac{\partial L_B}{\partial \mathbf{w}_k} \\
\mathbf{w}'_k &:= \mathbf{w}'_k - \lambda \frac{\partial L_B}{\partial \mathbf{w}'_k}
\end{aligned}
$$

The total gradient is a sum of local gradients:

$$\frac{\partial L_B}{\partial \mathbf{w}_k} = \frac{1}{N_B} \sum_{\{c,t\}_B} \frac{\partial l_{ct}}{\partial \mathbf{w}_k}$$

$$\frac{\partial l_{ct}}{\partial \mathbf{w}_k} = -\frac{\partial}{\partial \mathbf{w}_k} \left\{ \mathbf{w}_c^T \mathbf{w'}_t \right\} + \frac{\partial}{\partial \mathbf{w}_k} \log\left( \sum_i \exp\left( \mathbf{w}_c^T \mathbf{w}'_i \right) \right)$$

$$\frac{\partial l_{ct}}{\partial \mathbf{w}'_k} = -\frac{\partial}{\partial \mathbf{w}'_k} \left\{ \mathbf{w}_c^T \mathbf{w'}_t \right\} + \frac{\partial}{\partial \mathbf{w}'_k} \log\left( \sum_i \exp\left( \mathbf{w}_c^T \mathbf{w}'_i \right) \right)$$

# CBOW - Continuous Bag-of-Word

***1. <u>One word context</u>***

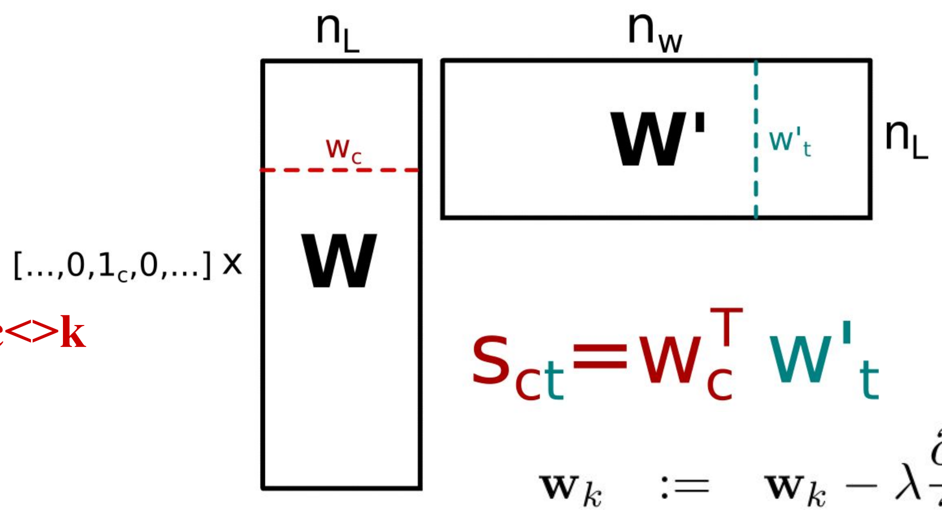
**“the math”**

Gradients have particular properties:

$$\boxed{\frac{\partial l_{ct}}{\partial \mathbf{w}_k} = -\frac{\partial}{\partial \mathbf{w}_k}\left\{\mathbf{w}_c^T \mathbf{w'}_t\right\} + \frac{\partial}{\partial \mathbf{w}_k}\log\left(\sum_i \exp\left(\mathbf{w}_c^T \mathbf{w}'_i\right)\right)}$$

=0 if **c<>k**

$$\frac{\partial l_{ct}}{\partial \mathbf{w}'_k} = -\frac{\partial}{\partial \mathbf{w}'_k}\left\{\mathbf{w}_c^T \mathbf{w'}_t\right\} + \frac{\partial}{\partial \mathbf{w}'_k}\log\left(\sum_i \exp\left(\mathbf{w}_c^T \mathbf{w}'_i\right)\right)$$




$$S_{ct} = w_c^T \, w'_t$$

$$\mathbf{w}_k := \mathbf{w}_k - \lambda \frac{\partial L_B}{\partial \mathbf{w}_k}$$

$$\mathbf{w}'_k := \mathbf{w}'_k - \lambda \frac{\partial L_B}{\partial \mathbf{w}'_k}$$

**W** is updated only at one row which corresponds to the context word.

*Consider case when some word appeared e.g. 10 times and $n_L$=100…*

**W’** is updated for every word in the vocabulary.

*Consider vocabulary of size $10^6$….*

**We will tackle computational improvements in the nexts slides.**

# CBOW - Continuous Bag-of-Word

## 1. *One word context*

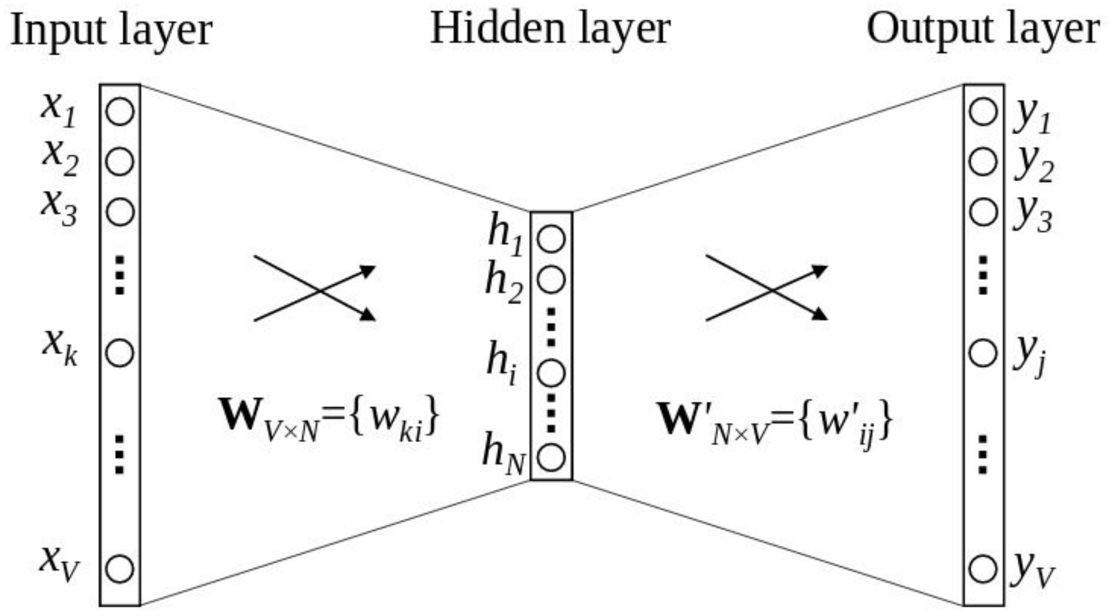



Figure 1: A simple CBOW model with only one word in the context

**"the quick brown fox jumped over the lazy dog"**

- We start from creating context/target pairs: (**the** : **quick**, **quick** : **brown**, **brown** : **fox**, ...)
- The words are converted to sparse orthogonal representation: (**1** : **2**, **2** : **3**, **3** : **4**, ...)

*This is what we know so far... any question?*

# CBOW - Continuous Bag-of-Word

***2. Multi word context***

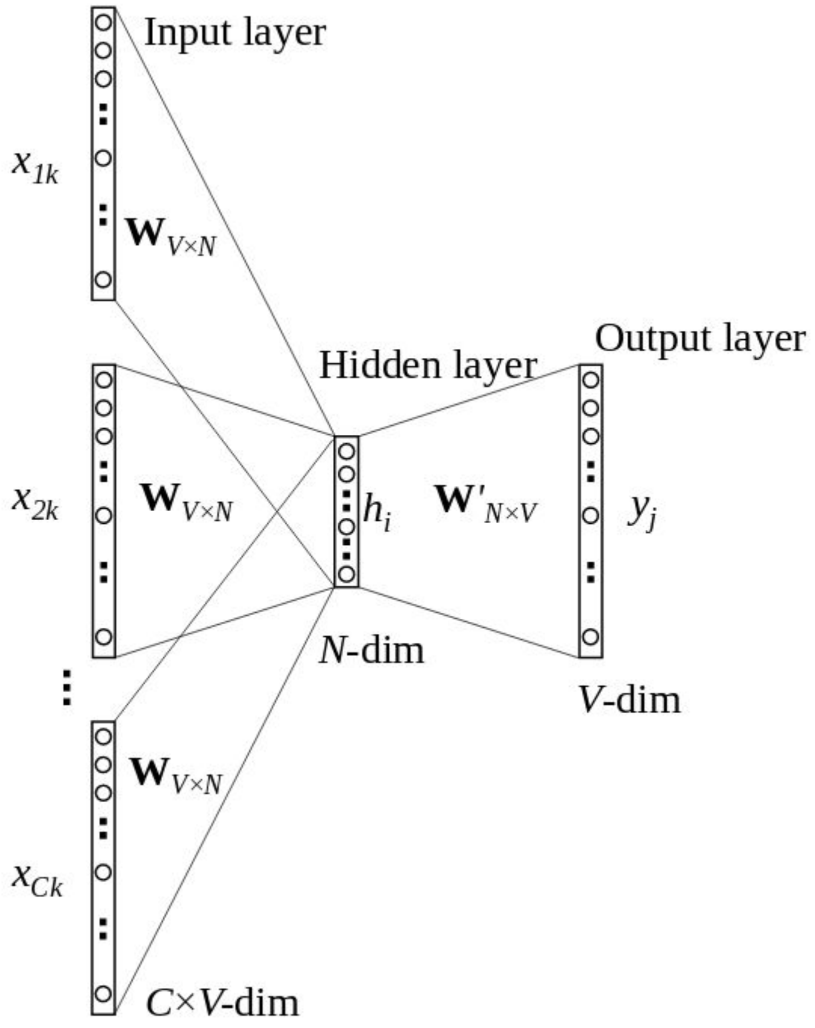



Figure 2: Continuous bag-of-word model

**"the quick brown fox jumped over the lazy dog"**

- We start from creating context/target groups: (**the, brown** : **quick**, **quick, fox** : **brown**, **brown, jumped** : **fox**, ...)
- The words are converted to sparse orthogonal representation: (**1,3** : **2**, **2,4** : **3**, **3,5** : **4**, ...)

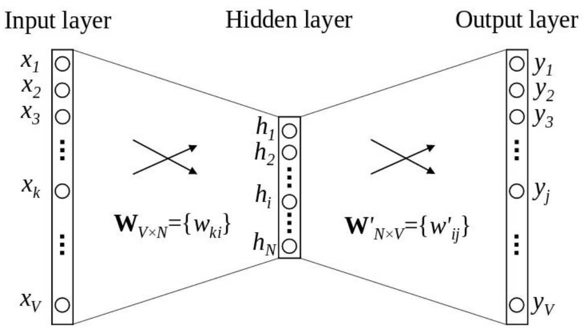



Figure 1: A simple CBOW model with only one word in the context

one word context

# CBOW - Continuous Bag-of-Word

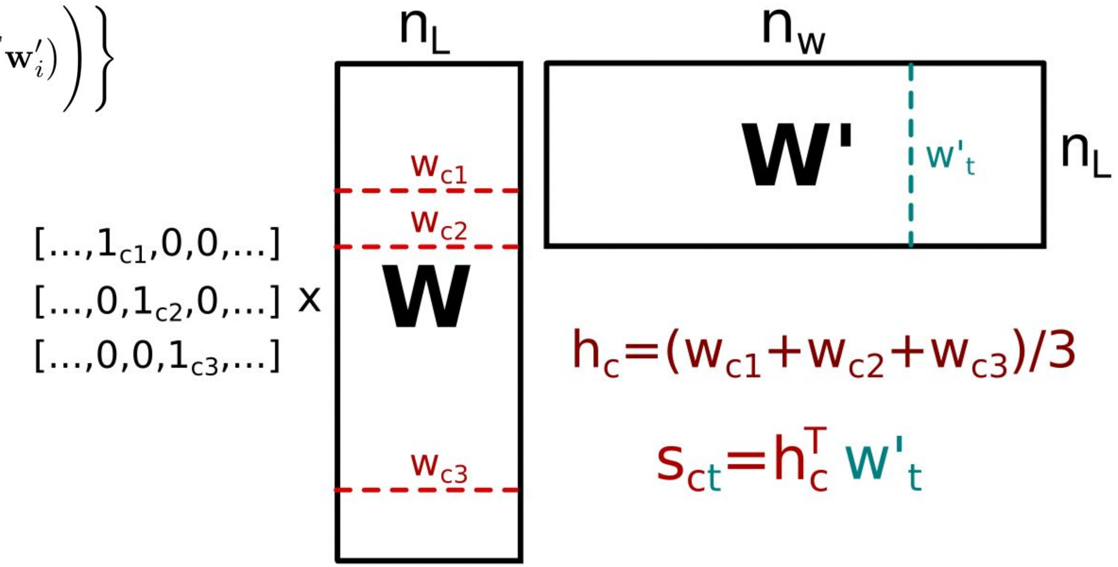
## 2. Multi word context

**"the math was:** $L_B = \frac{1}{N_B} \sum_{\{c,t\}_B} \left\{ -\mathbf{w}_c^T \mathbf{w}'_t + \log \left( \sum_i \exp\left(\mathbf{w}_c^T \mathbf{w}'_i\right) \right) \right\}$

$$L_B = \frac{1}{N_B} \sum_{\{c,t\}_B} \left\{ -\mathbf{h}_c^T \mathbf{w}'_t + \log \left( \sum_i \exp\left(\mathbf{h}_c^T \mathbf{w}'_i\right) \right) \right\}$$

$$\mathbf{h}_c = \frac{1}{N_c} \sum_{i \in c} \mathbf{w}_{c_i}$$

$h_c$ is an average of context words **c**




$$h_c=(w_{c1}+w_{c2}+w_{c3})/3$$

$$s_{ct}=h_c^T w'_t$$

$$\mathbf{w}_k := \mathbf{w}_k - \lambda \frac{\partial L_B}{\partial \mathbf{w}_k}$$

$$\mathbf{w}'_k := \mathbf{w}'_k - \lambda \frac{\partial L_B}{\partial \mathbf{w}'_k}$$

*target word*

**"the quick brown fox jumped over the lazy dog"**

*context window size = 1*

**Note:**

Same behaviour while computing gradients but **$N_c$** context words is now affected during backpropagation

# CBOW - Continuous Bag-of-Word

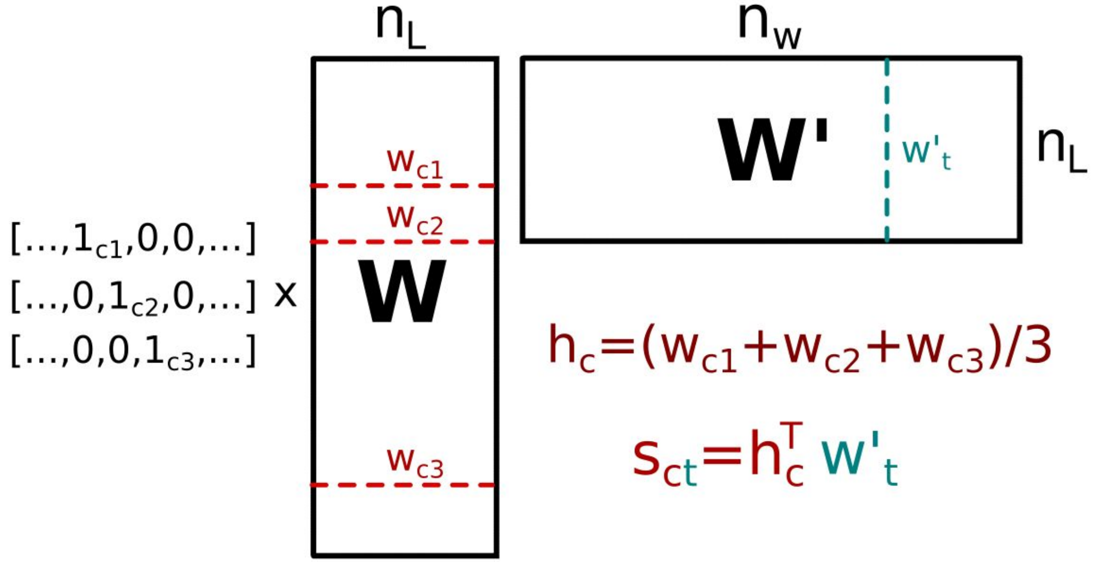
## 2. Multi word context

**"the math remains the same"**

$$L_B = \frac{1}{N_B} \sum_{\{c,t\}_B} \left\{ -\mathbf{h}_c^T \mathbf{w}'_t + \log \left( \sum_i \exp \left( \mathbf{h}_c^T \mathbf{w}'_i \right) \right) \right\}$$

$$\mathbf{h}_c = \frac{1}{N_c} \sum_{i \in c} \mathbf{w}_{c_i}$$

$h_c$ is an average of context words **c**




Code remain almost the same:

```
c=[2,3,0]  # context words
t=4  # target word

score_ct = lambda c,t : W[c,:].mean(axis=0) @ Wp[:,t]
scores_c = [ score_ct(c, i) for i in range(10) ]

prob_ct = exp(score_ct(c, t))/sum(exp(scores_c))

loss_ct = -log(prob_ct)
```

# CBOW - any questions???

***2. Multi word context***

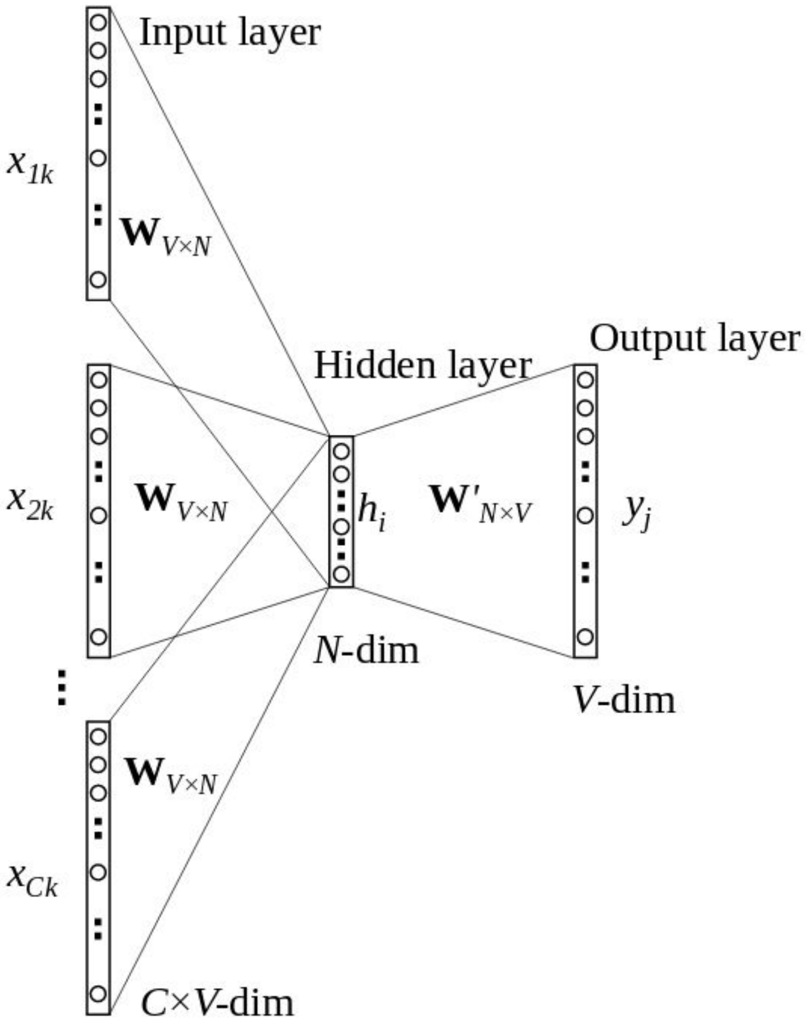



Figure 2: Continuous bag-of-word model

"the quick brown fox jumped over the lazy dog"

- We start from creating context/target groups: (**the, brown** : **quick**, **quick, fox** : **brown**, **brown, jumped** : **fox**, ...)
- The words are converted to sparse orthogonal representation: (**1,3** : **2**, **2,4** : **3**, **3,5** : **4**, ...)

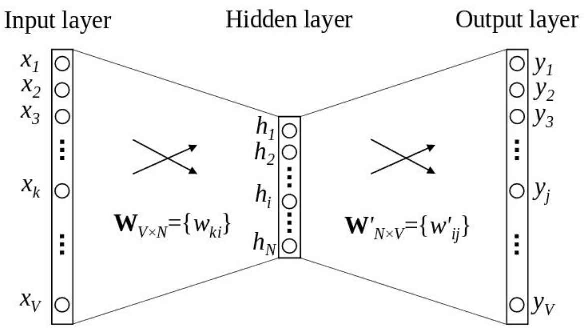



Figure 1: A simple CBOW model with only one word in the context

one word context

# CBOW and Skip-Gram Model

**2. Multi word context**

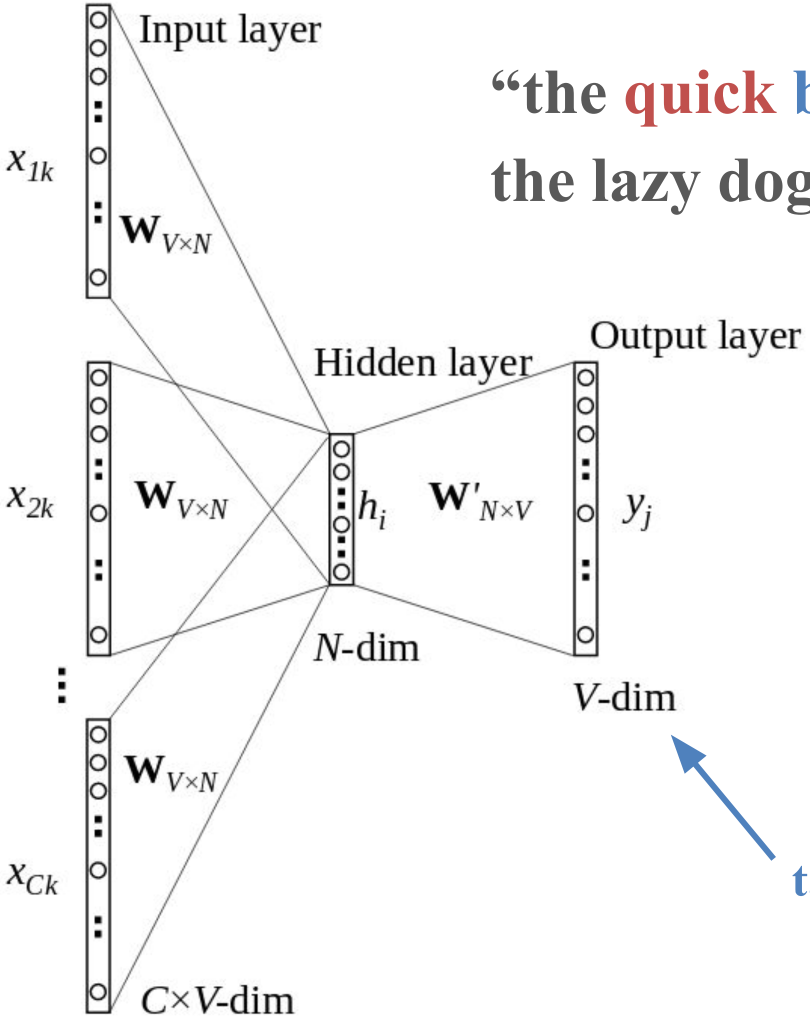



Figure 2: Continuous bag-of-word model

**3. Skip Gram Model**

**"the quick brown fox jumped over the lazy dog"**

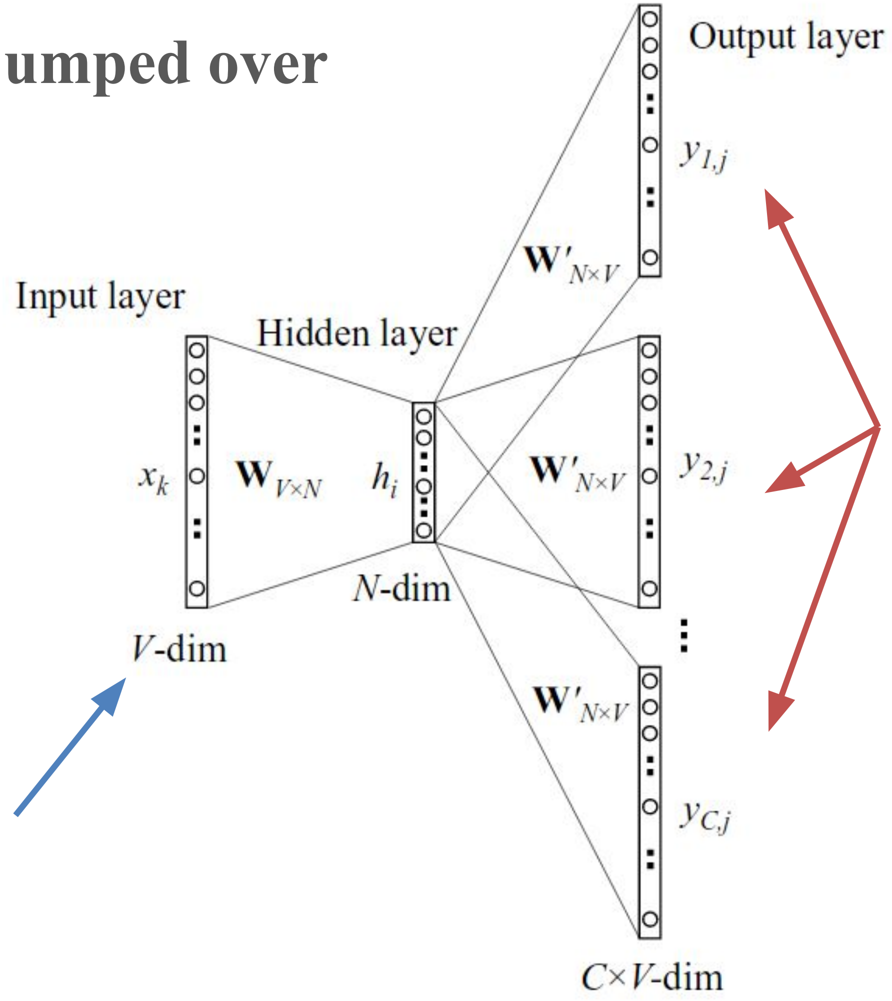



Figure 3: The skip-gram model.

target word

context words

# Skip-Gram Model

## "the math remains the same"

For a given context/target pair we compute score

$$s_{c_i t} = \mathbf{w}_t^T \mathbf{w}'_{c_i}$$

and probability:

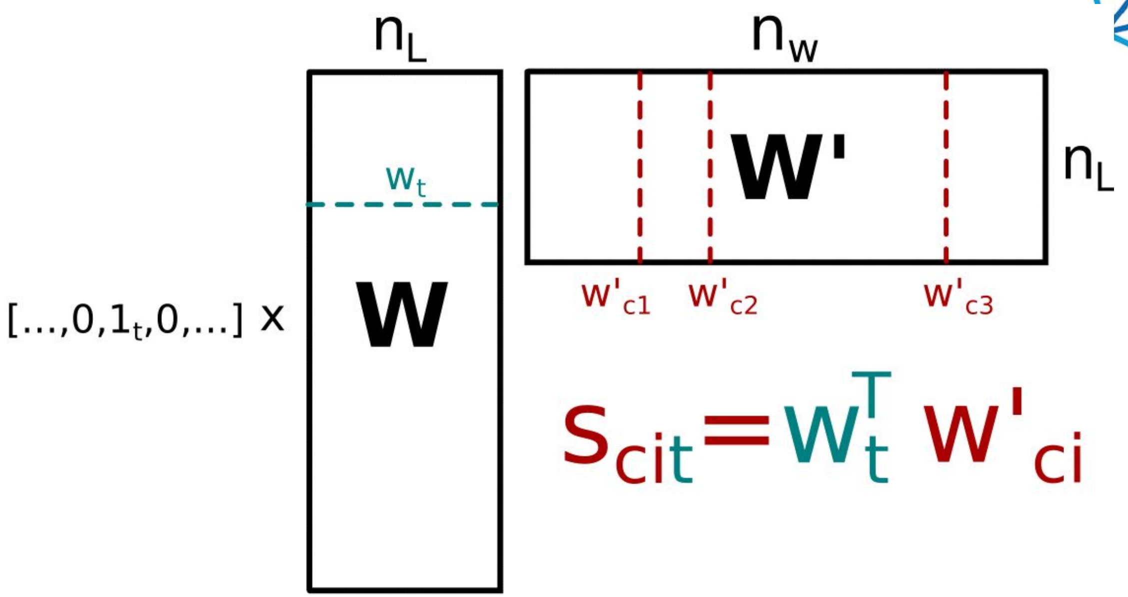
$$p_{c_i t} = \frac{\exp\left(s_{c_i t}\right)}{\sum_i \exp\left(s_{it}\right)}$$

Total probability of observing all context words given target is then

$$p_{ct} = p_{c_1 t} p_{c_2 t} \cdots p_{c_N t} = \prod_{j \in c} \frac{\exp\left(s_{c_j t}\right)}{\sum_i \exp\left(s_{it}\right)}$$

With loss

$$\begin{aligned} l_{ct} &= -\log\left(p_{ct}\right) = \\ &= -\sum_j s_{c_j t} + N_c \log\left(\sum_i \exp\left(s_{it}\right)\right) \end{aligned}$$




With loss per mini-batch

$$L_B = \frac{1}{N_B} \sum_{\{c,t\}_B} \left\{ -\sum_j \mathbf{w}_t^T \mathbf{w}'_{c_j} + N_c \log\left(\sum_i \exp\left(\mathbf{w}_t^T \mathbf{w}'_i\right)\right) \right\}$$

**Note:** Gradients will have similar form but a little different behaviour

# Skip-Gram Model

## “the math remains the same”

For a given context/target pair we compute score

$$s_{c_i t} = \mathbf{w}_t^T \mathbf{w}'_{c_i}$$

and probability:

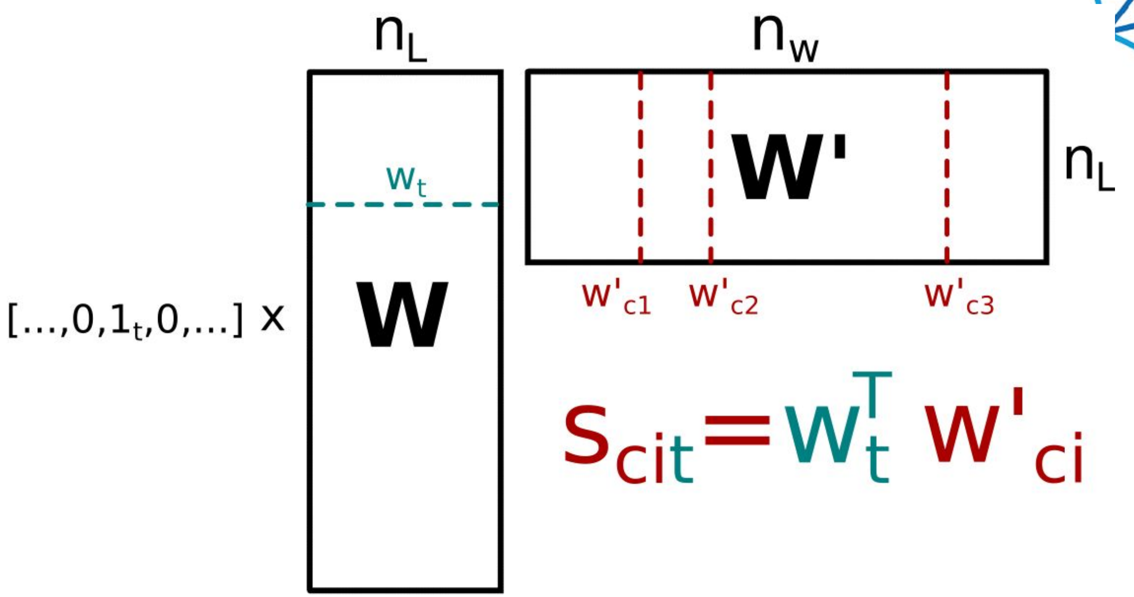
$$p_{c_i t} = \frac{\exp\left(s_{c_i t}\right)}{\sum_i \exp\left(s_{it}\right)}$$

Total probability of observing all context words given target is then

$$p_{ct} = p_{c_1 t} p_{c_2 t} \cdots p_{c_N t} = \prod_{j \in c} \frac{\exp\left(s_{c_j t}\right)}{\sum_i \exp\left(s_{it}\right)}$$

With loss

$$\begin{aligned} l_{ct} &= -\log\left(p_{ct}\right) = \\ &= -\sum_j s_{c_j t} + N_c \log\left(\sum_i \exp\left(s_{it}\right)\right) \end{aligned}$$




```
c=[2,3,0]  # context words -> W' matrix
t=4  # target word -> W matrix

score_ct = lambda c,t : W[t,:] @ Wp[:,c]
scores_t = [ score_ct(i, t) for i in range(10) ]

prob_ct = lambda c,t : exp(score_ct(c, t))/sum(exp(scores_c))
prob_t = [prob_ct(c_i, t) for c_i in c]

prob_tc = np.prod(prob_t)
loss_ct = -log(prob_tc)
```

# CBOW

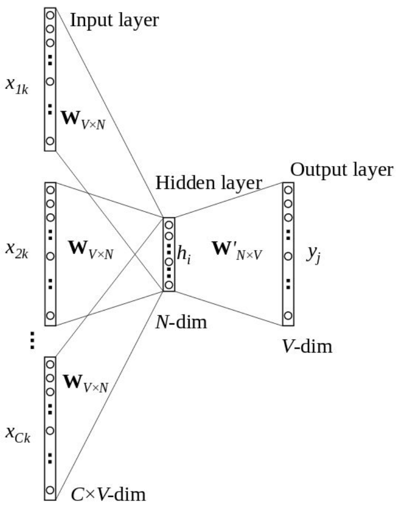
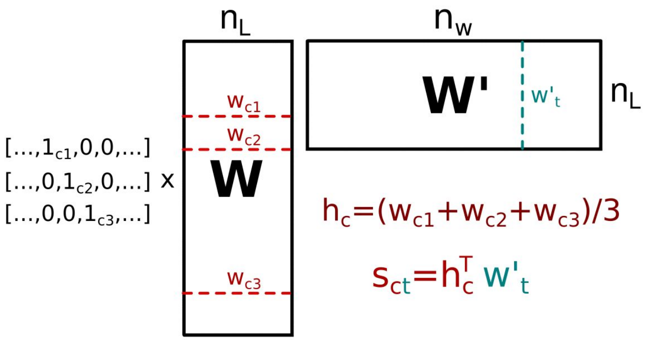



Figure 2: Continuous bag-of-word model




$h_c=(w_{c1}+w_{c2}+w_{c3})/3$

$s_{ct}=h_c^T\, w'_t$

$$L_B = \frac{1}{N_B}\sum_{\{c,t\}_B}\left\{-\mathbf{h}_c^T\mathbf{w}'_t + \log\left(\sum_i \exp\left(\mathbf{h}_c^T\mathbf{w}'_i\right)\right)\right\}$$

$$\mathbf{h}_c = \frac{1}{N_c}\sum_{i\in c}\mathbf{w}_{c_i}$$

- smoothes context, so average out statistical information
- faster to train
- better in accuracy for frequent words

- Inefficient usage of statistics
- Can capture complex patterns beyond word similarity

# Skip-Gram

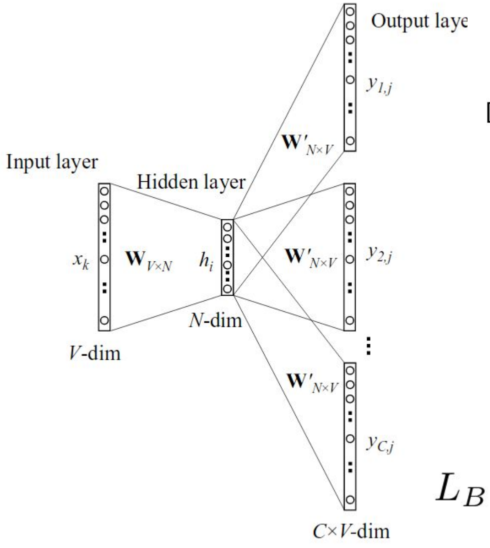
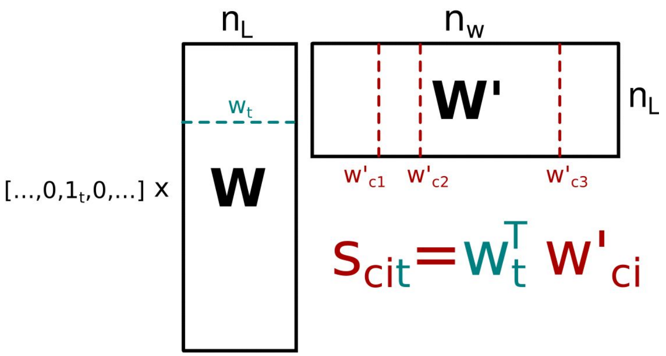



Figure 3: The skip-gram model.




$s_{cit}=w_t^T\, w'_{ci}$

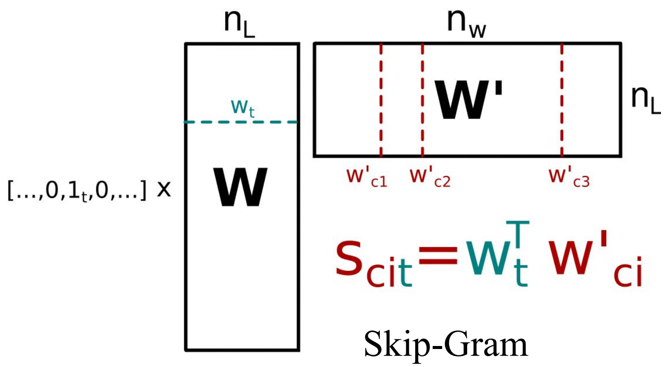
$$p_{ct} = p_{c_1t}p_{c_2t}\cdots p_{c_Nt} = \prod_{j\in c}\frac{\exp\left(s_{c_jt}\right)}{\sum_i \exp\left(s_{it}\right)}$$

$$L_B = \frac{1}{N_B}\sum_{\{c,t\}_B}\left\{-\sum_j \mathbf{w}_t^T\mathbf{w}'_{c_j} + N_c\log\left(\sum_i \exp\left(\mathbf{w}_t^T\mathbf{w}'_i\right)\right)\right\}$$

- no smoothing effect, should give better results than CBOW
- slower to train
- better in accuracy for rare words

- Inefficient usage of statistics
- Can capture complex patterns beyond word similarity

# word2vec final question?




Skip-Gram

The models are trained until converge

$$\mathbf{W} \;:=\; \mathbf{W} - \lambda \frac{\partial L_B}{\partial \mathbf{W}}$$

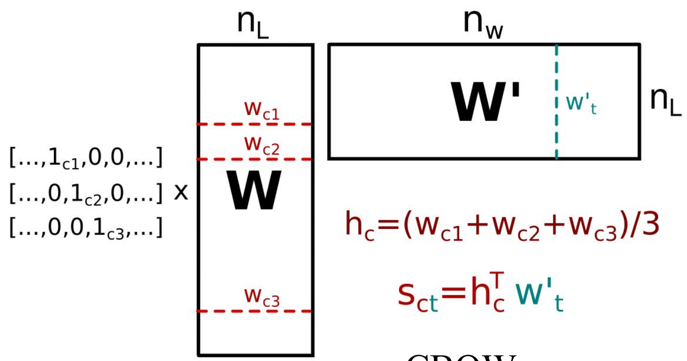
$$\mathbf{W}' \;:=\; \mathbf{W}' - \lambda \frac{\partial L_B}{\partial \mathbf{W}'}$$

Which matrix take to represent the words embeddings???

This will depend on implementation, but good solution is to take sum of both matrices

$$\mathbf{W}_{final} = \mathbf{W} + \mathbf{W}'^T$$

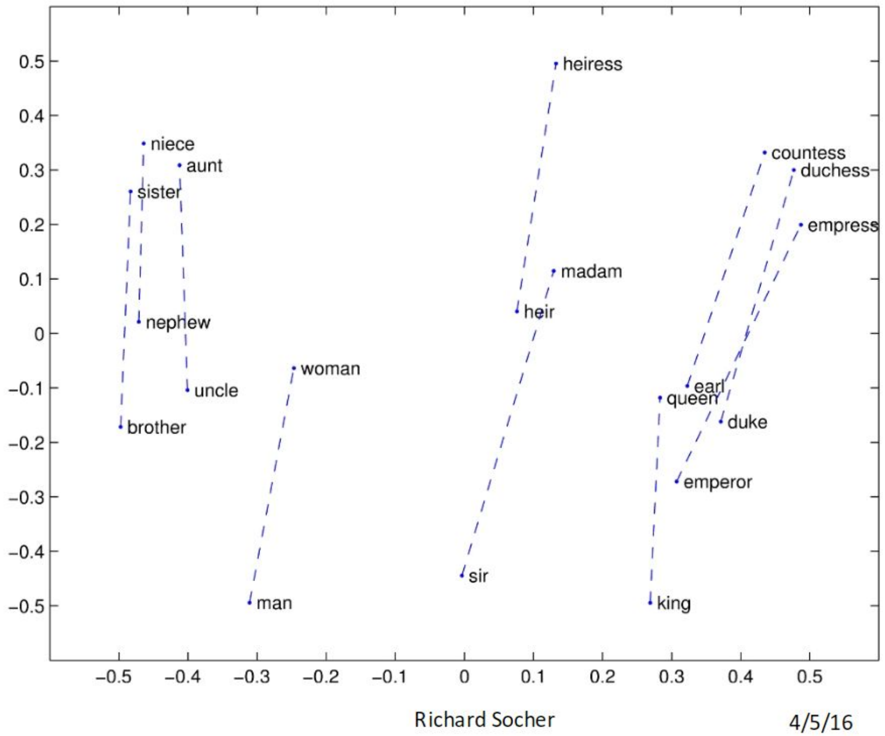
*however this can be done arbitrarily (there is no universal recipe for doing this)*




CBOW

# word2vec evaluation

**The task is non trivial (unsupervised learning):** There are special datasets which can be used for model evaluation




- **Testing words analogies:**

: gram4-superlative
bad worst big biggest
bad worst bright brightest
bad worst cold coldest
bad worst cool coolest
bad worst dark darkest
bad worst easy easiest
bad worst fast fastest
bad worst good best
bad worst great greatest

: gram7-past-tense
dancing danced decreasing decreased
dancing danced describing described
dancing danced enhancing enhanced
dancing danced falling fell
dancing danced feeding fed
dancing danced flying flew
dancing danced generating generated
dancing danced going went
dancing danced hiding hid
dancing danced hitting hit

- **Vector compositionality using element-wise addition**

| Czech + currency | Vietnam + capital | German + airlines | Russian + river | French + actress |
|---|---|---|---|---|
| koruna | Hanoi | airline Lufthansa | Moscow | Juliette Binoche |
| Check crown | Ho Chi Minh City | carrier Lufthansa | Volga River | Vanessa Paradis |
| Polish zolty | Viet Nam | flag carrier Lufthansa | upriver | Charlotte Gainsbourg |
| CTK | Vietnamese | Lufthansa | Russia | Cecile De |

***from Skip-Gram paper**: Distributed Representations of Words and Phrases and their Compositionality*

**Evaluation data sets can be found at:**

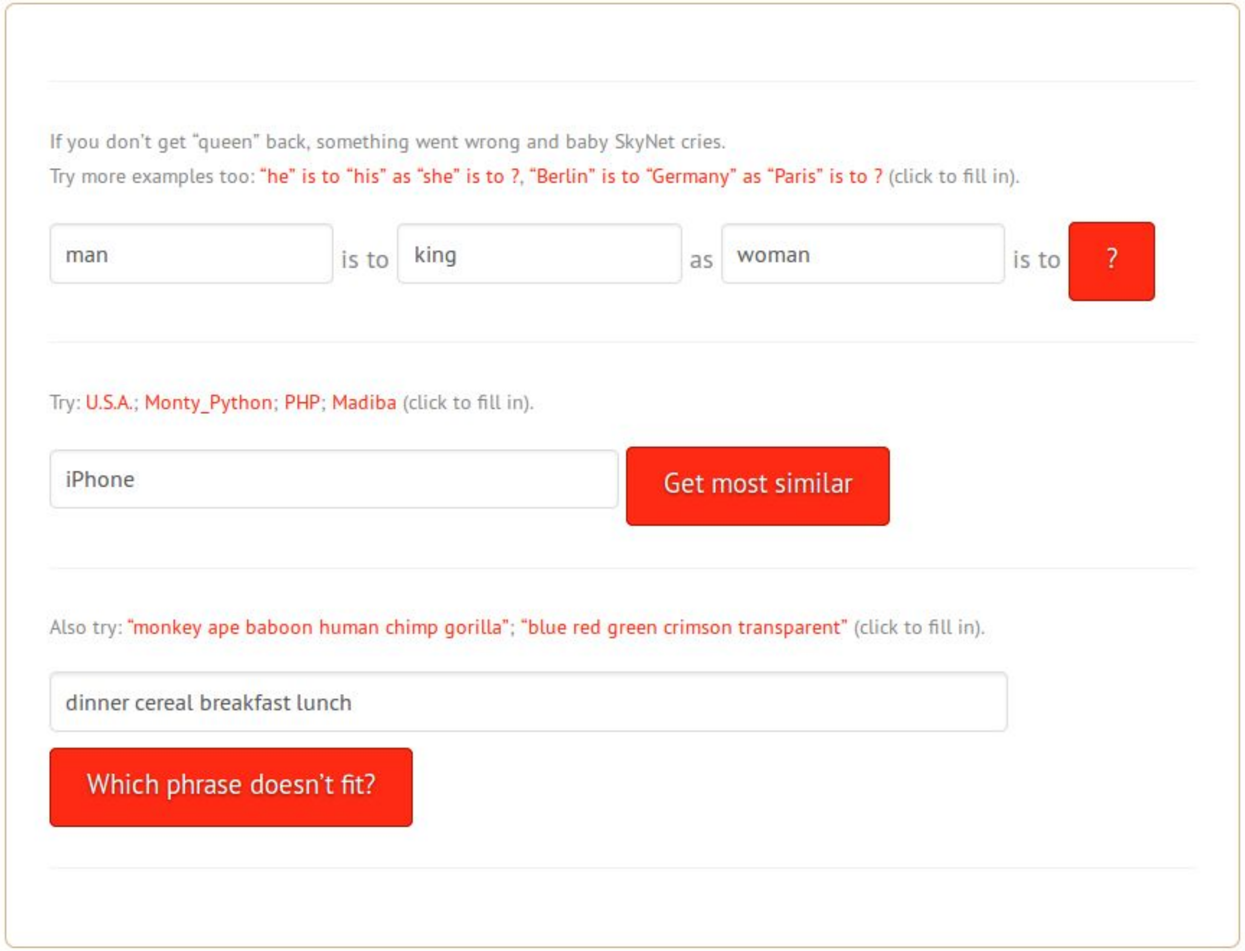
**word2vec / questions-words.txt**

**word2vec / questions-phrases.txt**

# word2vec - the fun part

## Time for fun!

https://rare-technologies.com/word2vec-tutorial/



# Improving speed and accuracy

$$\mathbf{W} := \mathbf{W} - \lambda \frac{\partial L_B}{\partial \mathbf{W}}$$

$$\mathbf{W}' := \mathbf{W}' - \lambda \frac{\partial L_B}{\partial \mathbf{W}'}$$

A. **Frequent words** will dominate gradients e.g. word **"the"** will appear before almost every noun: **the** cat, **the** dog.

**solution** (pre processing of the document with sentences)**:**

- remove super frequent words from corpus before training
- subsample frequent words before training :

**probability** that word $\mathbf{w_i}$ will be rejected from training set

$$P(w_i) = 1 - \sqrt{\frac{t}{f(w_i)}}$$

**chosen threshold** (hyperparameter)

**word frequency**

B. **Frequent phrases**: some words occur naturally in pairs, triples etc: *New York Times, Microsoft Office*, ...

where phrases are formed based on the unigram and bigram counts, using

**phrase threshold** (hyperparameter)

$$\text{score}(w_i, w_j) = \frac{\text{count}(w_i w_j) - \delta}{\text{count}(w_i) \times \text{count}(w_j)}. \qquad (6)$$

The $\delta$ is used as a discounting coefficient and prevents too many phrases consisting of very infrequent words to be formed. The bigrams with score above the chosen threshold are then used as phrases. Typically, we run 2-4 passes over the training data with decreasing threshold value, allowing longer phrases that consists of several words to be formed. We evaluate the quality of the phrase

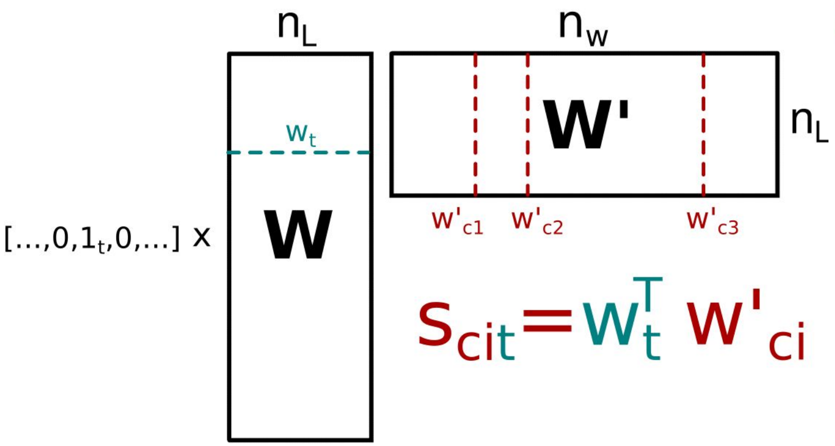
# Improving **speed** and accuracy

## 1. Negative Sampling (NEG) - Skip Gram case

Computing loss and its gradient is expensive operation because of the normalization term

$$p_{ct} = \prod_{j \in c} \frac{\exp\left(s_{c_j t}\right)}{\boxed{\sum_i \exp\left(s_{it}\right)}} = \prod_{j \in c} p_{c_j t}$$

$$l_{ct} = -\log\left(\prod_{j \in c} p_{c_j t}\right) = -\sum_{j \in c} \log\left(p_{c_j t}\right)$$

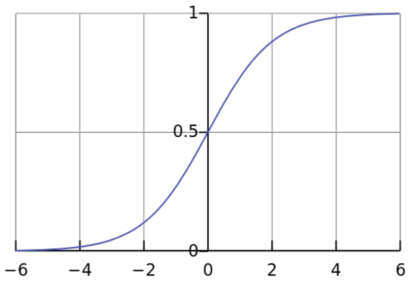



The exact log probability is replaced with approximated expression:

**Sigmoid function:**

$$\sigma(u) = \frac{1}{1+e^{-u}}$$

$$\sigma(-u) = 1 - \sigma(u)$$




$$\log\left(p_{c_j t}\right) = \log\left(\frac{\exp\left(s_{c_j t}\right)}{\sum_i \exp\left(s_{it}\right)}\right)$$

$$\approx \log\left(\sigma\left(\mathbf{w}_t^T \mathbf{w}'_{c_j}\right)\right) + \sum_{i \in w_{\text{neg}}}^{k} \log\left(1 - \sigma\left(\mathbf{w}_t^T \mathbf{w}'_i\right)\right)$$

**the good:** k ~ 30-40 hence dramatic reduction of computational complexity,
**the bad:** probability is not normalized
**the ugly:** never sure if it will work

**Case study:**

**t=cat**

**c={dog, mouse, horse}**

**$w_{neg}$={animal, door, house, elephant}**

***Note:*** *the quality of the learning will depend on sampled words*

# Improving **speed** and accuracy

## 2. Hierarchical Softmax

Standard Softmax function

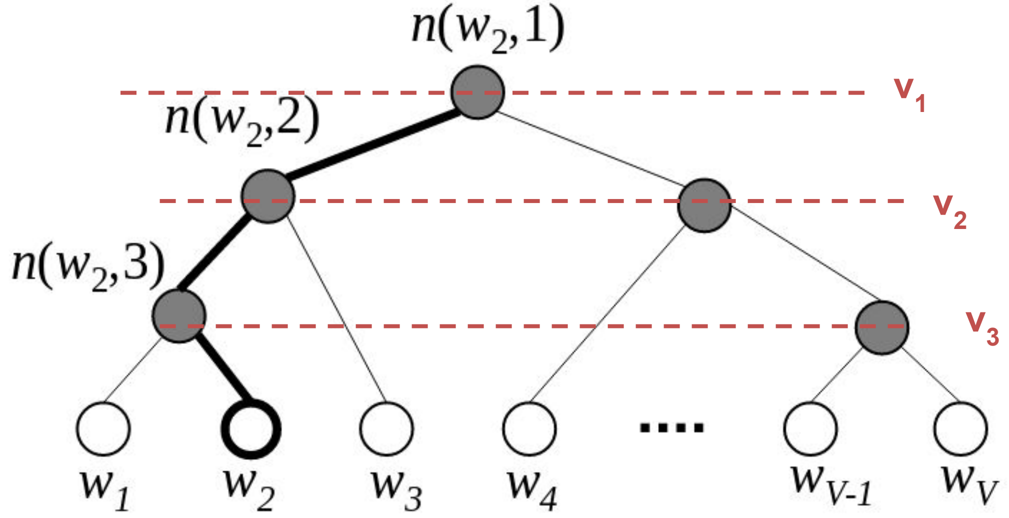
$$p_{c_i t} = \frac{\exp\left(s_{c_i t}\right)}{\sum_i \exp\left(s_{it}\right)}$$

Each word is positioned by its index in matrices **W** and **W'**

$$s_{c_i t} = \mathbf{w}_t^T \mathbf{w}'_{c_i}$$

Hierarchical Softmax treats words in matrix **W'** by leafs of Huffman binary tree

The probability of observing word $\mathbf{w_c}$ given $\mathbf{w_t}$ is given then by

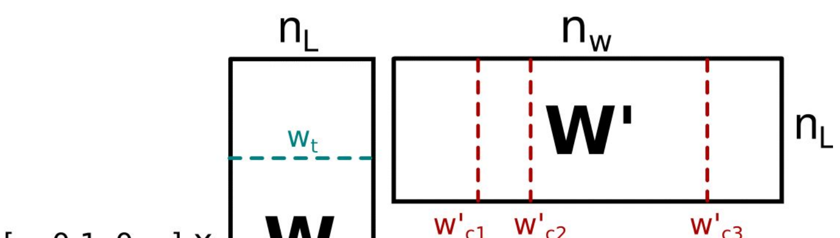



$$\begin{aligned} p_{ct} &= p(\mathbf{w}_c | \mathbf{w}_t) \\ &= \prod_{j=1}^{L(\mathbf{w}_c)-1} \sigma\left([\![ n(\mathbf{w}_c, j+1) = \mathrm{ch}\left(n\left(\mathbf{w}_c, j\right)\right)]\!] \cdot \mathbf{w}_t^T \mathbf{v}_{n(\mathbf{w}_c, j)}\right) \end{aligned}$$

$$[\![x]\!] = \begin{cases} 1 & \text{if } x \text{ is true;} \\ -1 & \text{otherwise.} \end{cases}$$

$L\left(\mathbf{w_c}\right)$ - length of path for word $\mathbf{w_c}$

*Note: Huffman tree assigns short codes to frequent words*




$n\left(\mathbf{w}_c, j\right)$ - means the **j-th** unit on the path from root to the word **w**

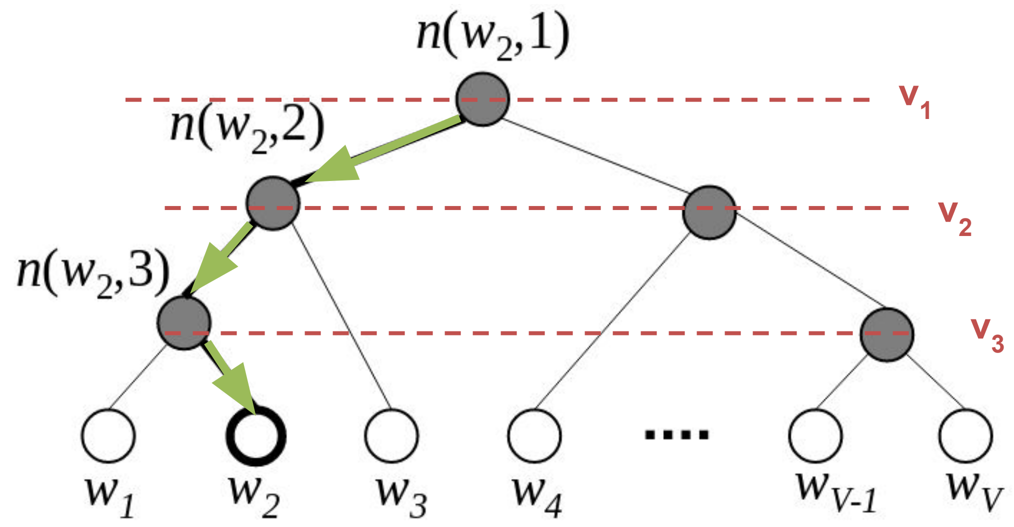
$\mathrm{ch}\left(\mathrm{n}\left(\mathbf{w}_c, \mathrm{j}\right)\right)$ - left child of unit n

# Improving **speed** and accuracy

## Hierarchical Softmax

The probability of observing word $\mathbf{w}_c$ given $\mathbf{w}_t$ is given then by

$$
\begin{aligned}
p_{ct} &= p(\mathbf{w}_c|\mathbf{w}_t) \\
&= \prod_{j=1}^{L(\mathbf{w}_c)-1} \sigma\left([n(\mathbf{w}_c, j+1) = \mathrm{ch}\left(n\left(\mathbf{w}_c, j\right)\right)] \cdot \mathbf{w}_t^T \mathbf{v}_{n(\mathbf{w}_c, j)}\right)
\end{aligned}
$$



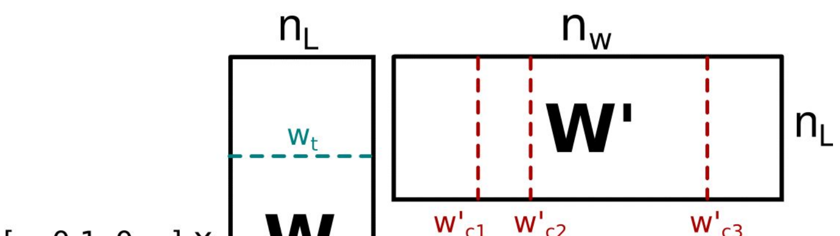

### Case study #1:

Given target word $\mathbf{w}_t$ we want to compute probability of observing context word $\mathbf{w}_c = \mathbf{w}_2$

- From tree structure we have: $L(\mathbf{w}_2)-1=3$
- Additionally,
  - $n(\mathbf{w}_c,1)=1$
  - $n(\mathbf{w}_c,2)=2$
  - $n(\mathbf{w}_c,3)=3$
  - $n(\mathbf{w}_c,4)=4$

$$
\begin{aligned}
p(\mathbf{w}_2|\mathbf{w}_t) &= \prod_{j=1}^{3} \sigma\left([\cdot] \cdot \mathbf{w}_t^T \mathbf{v}_{n(\mathbf{w}_2, j)}\right) \\
&= \sigma\left(\mathbf{w}_t^T \mathbf{v}_1\right) \sigma\left(\mathbf{w}_t^T \mathbf{v}_2\right) \sigma\left(-\mathbf{w}_t^T \mathbf{v}_3\right)
\end{aligned}
$$

*Note: Huffman tree assigns short codes to frequent words*

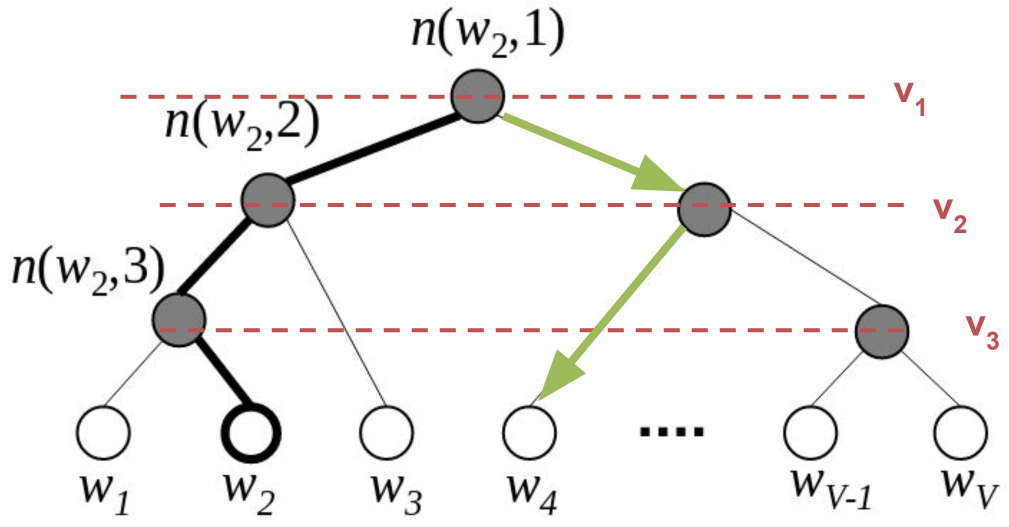


# Improving **speed** and accuracy

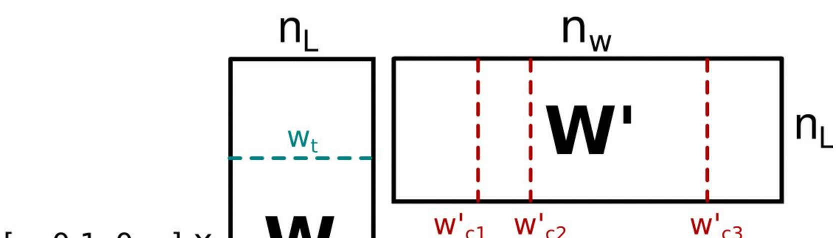
## Hierarchical Softmax

The probability of observing word $\mathbf{w}_c$ given $\mathbf{w}_t$ is given then by

$$\begin{aligned} p_{ct} &= p(\mathbf{w}_c|\mathbf{w}_t) \\ &= \prod_{j=1}^{L(\mathbf{w}_c)-1} \sigma\left([n(\mathbf{w}_c, j+1) = \mathrm{ch}\left(n\left(\mathbf{w}_c, j\right)\right)] \cdot \mathbf{w}_t^T \mathbf{v}_{n(\mathbf{w}_c, j)}\right) \end{aligned}$$




*Note: Huffman tree assigns short codes to frequent words*




## Case study #2:

Given target word $\mathbf{w}_t$ we want to compute probability of observing context word $\mathbf{w}_c = \mathbf{w}_4$

- From tree structure we have: $L(\mathbf{w}_4)-1=2$
- Additionally,
  - $n(\mathbf{w}_c,1)=1$
  - $n(\mathbf{w}_c,2)=2$
  - $n(\mathbf{w}_c,3)=4$

$$\begin{aligned} p(\mathbf{w}_4|\mathbf{w}_t) &= \prod_{j=1}^{2} \sigma\left([\cdot] \cdot \mathbf{w}_t^T \mathbf{v}_{n(\mathbf{w}_4, j)}\right) \\ &= \sigma\left(-\mathbf{w}_t^T \mathbf{v}_1\right) \sigma\left(\mathbf{w}_t^T \mathbf{v}_2\right) \end{aligned}$$

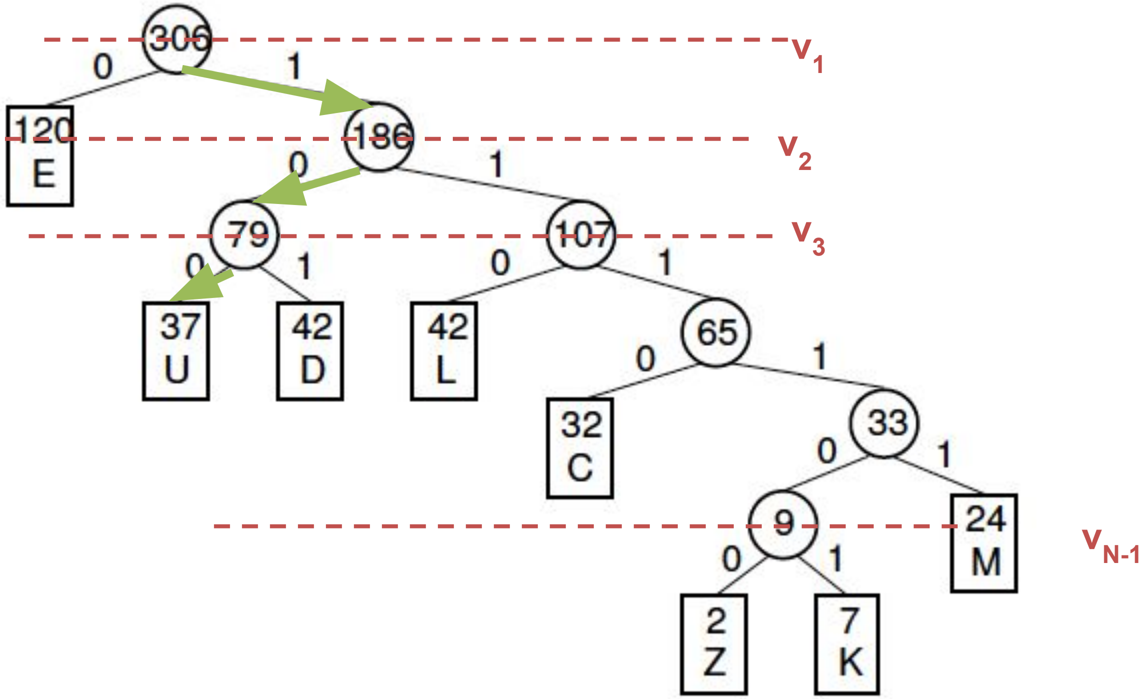
<u>**Note: Easier to compute than case #1**</u>

# Improving **speed** and accuracy

## Hierarchical Softmax

The probability of observing word $\mathbf{w_c}$ given $\mathbf{w_t}$ is given then by

$$p_{ct} = p(\mathbf{w}_c|\mathbf{w}_t) = \prod_{j=1}^{L(\mathbf{w}_c)-1} \sigma\left([n(\mathbf{w}_c, j+1) = \mathrm{ch}\left(n\left(\mathbf{w}_c, j\right)\right)] \cdot \mathbf{w}_t^T \mathbf{v}_{n(\mathbf{w}_c,j)}\right)$$




**Properties:**

- maximum number of inner units is $n_w$-1
- frequent words require less computational power (short code) - fast training
- average complexity is O(~log($n_w$)) instead O($n_w$)
- probability is normalized to one

$$p_t = \sum_c p(\mathbf{w}_c|\mathbf{w}_t) = 1$$

# Implementation in gensim package

## Hierarchical Softmax

red - not related with theory
blue - related with discussed theory

```
class gensim.models.word2vec.Word2Vec(
    sentences=None, // sentences of iterable
    size=100, // latent space size - dimensionality of vectors
    alpha=0.025, // learning rate
    window=5, // maximum size of the window during scan
    min_count=5, // ignore all words with total frequency lower than this (OPT)
    max_vocab_size=None, // limit RAM when building vocabulary
    sample=0.001, // threshold for downsampling frequent words (OPT)
    seed=1, // seed for random number generator
    workers=3, // parallelization - number of threads
    min_alpha=0.0001,
    sg=0, // By default (sg=0), CBOW is used. Otherwise (sg=1), skip-gram is employed
    hs=0, // if 1, hierarchical softmax, otherwise NEG (OPT)
    negative=5, // number of negative samples (OPT)
    cbow_mean=1, // if 0, use the sum, if 1 use mean for CBOW model
    hashfxn=<built-in function hash>, // hash function to use to randomly initialize weights
    iter=5, // number of iterations (epochs) over the corpus
    trim_rule=None, // vocabulary trimming rule
    sorted_vocab=1, // if 1 (default), sort the vocabulary by descending frequency before words indexing
    batch_words=10000 // target size (in words) for batches passed to each worker )
```

https://radimrehurek.com/gensim/models/word2vec.html

# Skip Gram paper evaluation

[The performance of various Skip-gram models on the word analogy test](https://arxiv.org/pdf/1301.3781.pdf)

| Method | Time [min] | Syntactic [%] | Semantic [%] | Total accuracy [%] |
|---|---|---|---|---|
| NEG-5 | 38 | 63 | 54 | 59 |
| NEG-15 | 97 | 63 | 58 | **61** |
| HS-Huffman | 41 | 53 | 40 | 47 |
| NCE-5 | 38 | 60 | 45 | 53 |
| The following results use $10^{-5}$ subsampling | | | | |
| NEG-5 | 14 | 61 | 58 | 60 |
| NEG-15 | 36 | 61 | 61 | **61** |
| HS-Huffman | 21 | 52 | 59 | 55 |

Table 1: Accuracy of various Skip-gram 300-dimensional models on the analogical reasoning task as defined in [8]. NEG-$k$ stands for Negative Sampling with $k$ negative samples for each positive sample; NCE stands for Noise Contrastive Estimation and HS-Huffman stands for the Hierarchical Softmax with the frequency-based Huffman codes.

| Method | Dimensionality | No subsampling [%] | $10^{-5}$ subsampling [%] |
|---|---|---|---|
| NEG-5 | 300 | 24 | 27 |
| NEG-15 | 300 | 27 | 42 |
| HS-Huffman | 300 | 19 | **47** |

Table 3: Accuracies of the Skip-gram models on the phrase analogy dataset. The models were trained on approximately one billion words from the news dataset.

Table 5: *Comparison of models trained for three epochs on the same data and models trained for one epoch. Accuracy is reported on the full Semantic-Syntactic data set.*

| Model | Vector Dimensionality | Training words | Accuracy [%] | | | Training time [days] |
|---|---|---|---|---|---|---|
| | | | Semantic | Syntactic | Total | |
| 3 epoch CBOW | 300 | 783M | 15.5 | 53.1 | 36.1 | 1 |
| 3 epoch Skip-gram | 300 | 783M | 50.0 | 55.9 | 53.3 | 3 |
| 1 epoch CBOW | 300 | 783M | 13.8 | 49.9 | 33.6 | 0.3 |
| 1 epoch CBOW | 300 | 1.6B | 16.1 | 52.6 | 36.1 | 0.6 |
| 1 epoch CBOW | 600 | 783M | 15.4 | 53.3 | 36.2 | 0.7 |
| 1 epoch Skip-gram | 300 | 783M | 45.6 | 52.2 | 49.2 | 1 |
| 1 epoch Skip-gram | 300 | 1.6B | 52.2 | 55.1 | 53.8 | 2 |
| 1 epoch Skip-gram | 600 | 783M | 56.7 | 54.5 | 55.5 | 2.5 |

https://arxiv.org/pdf/1301.3781.pdf

Reminder

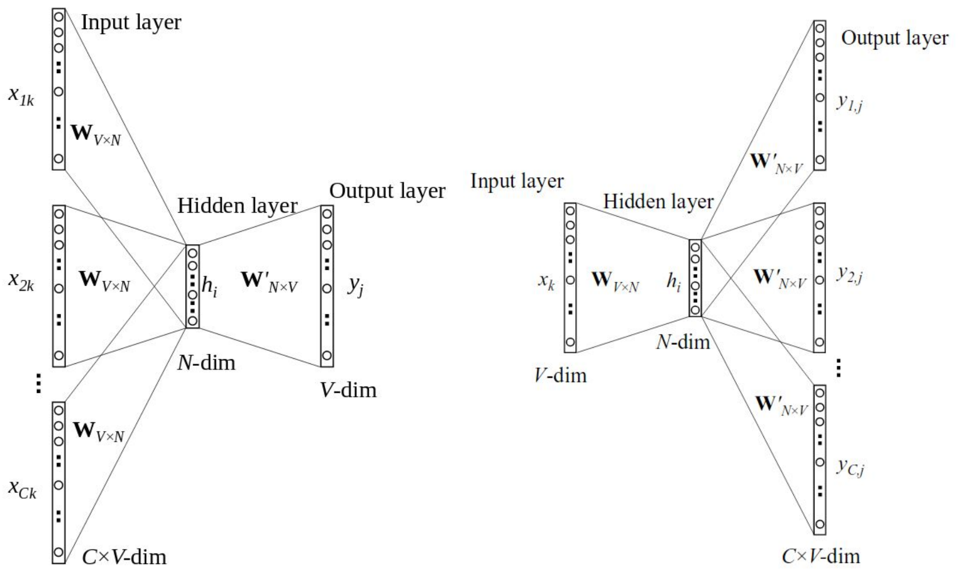



Figure 2: Continuous bag-of-word model

Figure 3: The skip-gram model.

# GloVe - Global vectors

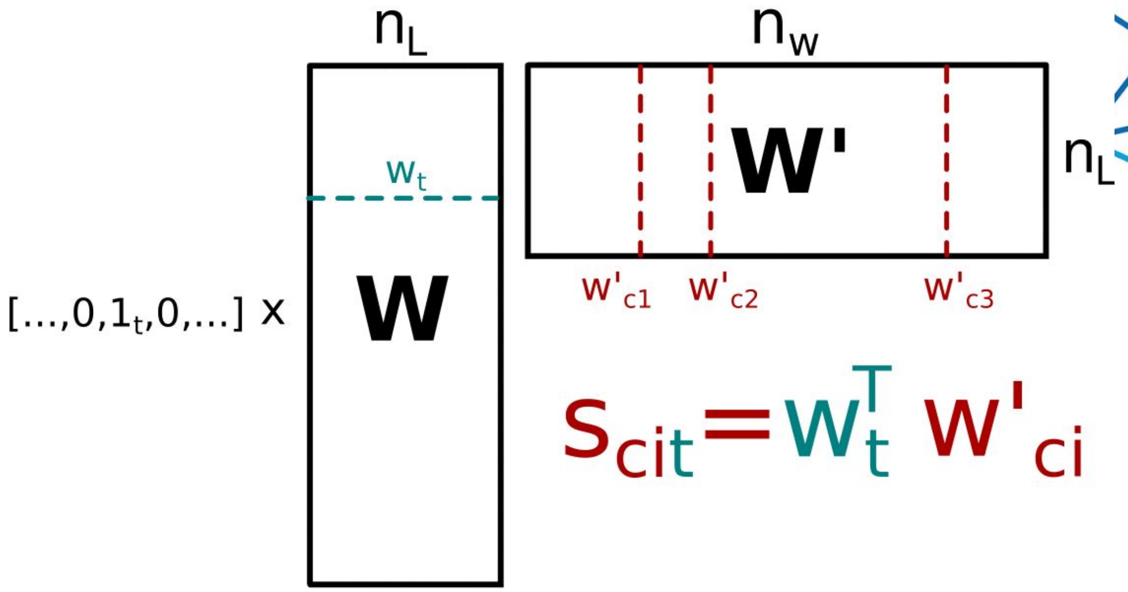
# GloVe - looking for analogies

For a given context/target pair we compute score

$$s_{c_i t} = \mathbf{w}_t^T \mathbf{w}'_{c_i}$$

and probability:

$$p_{c_i t} = \frac{\exp\left(s_{c_i t}\right)}{\sum_i \exp\left(s_{it}\right)}$$




$$p_{ct} = p_{c_1 t} p_{c_2 t} \cdots p_{c_N t} = \prod_{j \in c} \frac{\exp\left(s_{c_j t}\right)}{\sum_i \exp\left(s_{it}\right)}$$

This is what we **know** so far…

in general we have $p(\mathbf{w}_c|\mathbf{w}_t) = \prod_{j \in c} p(\mathbf{w}_{c_j}|\mathbf{w}_t)$ local loss $l(\mathbf{w}_c|\mathbf{w}_t) = -\log\left(\prod_{j \in c} p(\mathbf{w}_{c_j}|\mathbf{w}_t)\right) = -\sum_{j \in c} \log\left(p(\mathbf{w}_{c_j}|\mathbf{w}_t)\right)$

Total loss (loss over corpus) - first sum represent scan along corpus (**Skip-Gram model**)

$$L = -\sum_t \sum_{-m \leq j \leq m, j \neq 0} \log\left(p(\mathbf{w}_{t+j}|\mathbf{w}_t)\right)$$

# GloVe - looking for analogies

Total loss (loss over corpus) - first sum represent scan along corpus

$$L = -\sum_{t} \sum_{-m \leq j \leq m, j \neq 0} \log\left(p(\mathbf{w}_{t+j} | \mathbf{w}_t)\right)$$

Gradient of this is approximated with SGD with mini-batches.

Consider an example

**the fox jumped over dog then fox jumped over cow then fox jumped over pigeon**

However we can look globally on this problem:

$$\underbrace{\textbf{the fox jumped}}_{A}\ \textbf{over dog}\ \underbrace{\textbf{then fox jumped}}_{B}\ \textbf{over cow}\ \underbrace{\textbf{then fox jumped}}_{C}\ \textbf{over pigeon}$$

and group all **"fox"** words to compute total contribution to loss from that word

$$\begin{aligned} l\,(\mathbf{fox}) \;=\; & -\log\left(p(\text{the}|\mathbf{fox})\right) - \log\left(p(\text{jumped}|\mathbf{fox})\right) && \text{A} \\ & -\log\left(p(\text{then}|\mathbf{fox})\right) - \log\left(p(\text{jumped}|\mathbf{fox})\right) && \text{B} \\ & -\log\left(p(\text{then}|\mathbf{fox})\right) - \log\left(p(\text{jumped}|\mathbf{fox})\right) && \text{C} \end{aligned}$$

This simplify problem a lot, now we have 3 operation instead of 6!

$$l\,(\mathbf{fox}) \;=\; -\log\left(p(\text{the}|\mathbf{fox})\right) - 3\log\left(p(\text{jumped}|\mathbf{fox})\right) - 2\log\left(p(\text{then}|\mathbf{fox})\right)$$

# GloVe - looking for analogies

Total loss (loss over corpus) - first sum represent scan along corpus

$$L = -\sum_{t} \sum_{-m \leq j \leq m, j \neq 0} \log \left( p(\mathbf{w}_{t+j} | \mathbf{w}_t) \right)$$

Gradient of this is approximated with SGD with mini-batches.

**the fox jumped over dog then fox jumped over cow then fox jumped over pigeon**

$$l\left(\mathbf{fox}\right) = -\log\left(p(\mathrm{the}|\mathbf{fox})\right) - 3\log\left(p(\mathrm{jumped}|\mathbf{fox})\right) - 2\log\left(p(\mathrm{then}|\mathbf{fox})\right)$$

In general we can write loss for **fox** word using **co-occurrence** matrix

$$l\left(\mathbf{fox}\right) = -\sum_{w_i} O\left(\mathrm{fox}, w_i\right) \log\left(p(w_i|\mathbf{fox})\right)$$

where **O** is the co-occurrence matrix eg:

$$\begin{aligned} O\left(\mathrm{fox}, \mathrm{jumped}\right) &= 3 \\ O\left(\mathrm{fox}, \mathrm{cow}\right) &= 0 \end{aligned}$$

Finally **total loss** can be expressed in terms of co-occurrence matrix

$$L = \sum_{w_t} l\left(w_t\right) = -\sum_{w_t} \sum_{w_i} O\left(w_t, w_i\right) \log\left(p(w_i|\mathrm{w}_t)\right)$$

using simpler notation
with V being vocabulary size

$$L \equiv -\sum_{i}^{V} \sum_{j}^{V} O_{ij} \log\left(p_{ij}\right)$$

**Note:** In order to compute co-occurrence matrix one has to scan all corpus once and compute statistics.

# GloVe - looking for analogies

Total loss (loss over vocabulary)

$$L \equiv -\sum_{i}^{V}\sum_{j}^{V} O_{ij} \log\left(p_{ij}\right)$$

w define new variable

$$O_i \equiv \sum_{k}^{V} O_{ik} \qquad \tilde{O}_{ij} \equiv \frac{O_{ij}}{O_i}$$

is normalized to 1

$$L = -\sum_{i}^{V}\frac{O_i}{O_i}\sum_{j}^{V} O_{ij}\log\left(p_{ij}\right) = -\sum_{i}^{V} O_i \sum_{j}^{V}\frac{O_{ij}}{O_i}\log\left(p_{ij}\right) = -\sum_{i}^{V} O_i \sum_{j}^{V}\tilde{O}_{ij}\log\left(p_{ij}\right) = -\sum_{i}^{V} O_i \mathbf{H}\left(\tilde{O}_i, p_i\right)$$

$$\mathbf{H}\left(\tilde{O}_i, p_i\right) \equiv \sum_{j}^{V}\tilde{O}_{ij}\log\left(p_{ij}\right)$$

**is definition of cross entropy!**

**Note: Skip-Gram model is a weighted sum of cross entropy error**

By minimizing cross entropy H we minimize the distance between true distribution **Q** and learned **p.**

$$\mathbf{H}\left(\tilde{O}_i, p_i\right) = \sum_{j}^{V}\tilde{O}_{ij}\log\left(\tilde{O}_{ij}\right) - \sum_{j}^{V}\tilde{O}_{ij}\log\left(\frac{\tilde{O}_{ij}}{p_{ij}}\right)$$

KL - divergence

# A theory behind t-SNE

**taken from t-SNE presentation!!**

- Very basic assumption:

We want to have $\mathbf{p_{ij}}$ to be equal to $\mathbf{q_{ij}}$ i.e. low dimensional vectors **Y** correctly model similarities between high-dimensional data **X**.

*but how to do it?*

$$p_{ij} = \frac{p_{j|i} + p_{i|j}}{2n} = q_{ij} = \frac{\left(1 + \|y_i - y_j\|^2\right)^{-1}}{\sum_{k \neq l}\left(1 + \|y_k - y_l\|^2\right)^{-1}}.$$

We can define a measure between two distributions which will tell as how close those distributions are:

$$C = KL(P||Q) = \sum_i \sum_j p_{ij} \log \frac{p_{ij}}{q_{ij}}.$$

**Kullback–Leibler** divergence

We are going to minimize **C** with respect to **Y** vectors

**Notes:**

- C is asymmetric
- $q_{ij}$ is force to be large as $p_{ij}$
- but $q_{ij}$ cannot be larger than one, due to normalization, hence stability

$\mathbf{p_{ij}}$ are computed once in the first stage of **t-SNE**

# GloVe - looking for analogies

Total loss (loss over vocabulary)

$$L \equiv -\sum_i^V \sum_j^V O_{ij} \log\left(p_{ij}\right)$$

w define new variable

$$O_i \equiv \sum_k^V O_{ik} \qquad \tilde{O}_{ij} \equiv \frac{O_{ij}}{O_i}$$

is normalized to 1

$$L = -\sum_i^V \frac{O_i}{O_i} \sum_j^V O_{ij} \log\left(p_{ij}\right) = -\sum_i^V O_i \sum_j^V \frac{O_{ij}}{O_i} \log\left(p_{ij}\right) = -\sum_i^V O_i \sum_j^V \tilde{O}_{ij} \log\left(p_{ij}\right) = -\sum_i^V O_i \mathbf{H}\left(\tilde{O}_i, p_i\right)$$

$$\mathbf{H}\left(\tilde{O}_i, p_i\right) = \sum_j^V \tilde{O}_{ij} \log\left(\tilde{O}_{ij}\right) - \sum_j^V \tilde{O}_{ij} \log\left(\frac{\tilde{O}_{ij}}{p_{ij}}\right) = \mathbf{H}\left(\tilde{O}_i\right) - \mathbf{KL}\left(\tilde{O}_i, p_i\right)$$

KL - divergence

$p_{ij}\left(\mathbf{W}, \mathbf{W}'\right)$ - only the learned probability depends on **W, W'**

$$L = -\sum_i^V O_i \mathbf{H}\left(\tilde{O}_i\right) + \sum_i^V O_i \mathbf{KL}\left(\tilde{O}_i, p_i\right)$$

$$\frac{\partial L}{\partial \mathbf{W}} = \frac{\partial}{\partial \mathbf{W}} \sum_{i,j}^V O_{ij} \log\left(\frac{\tilde{O}_{ij}}{p_{ij}}\right)$$

**t-SNE like loss !**

$$C = KL(P||Q) = \sum_i \sum_j p_{ij} \log \frac{p_{ij}}{q_{ij}}.$$

Yesss! We could use t-SNE solver for finding low dimensional word representations

# GloVe - relation to word2vec

http://nlp.stanford.edu/pubs/glove.pdf

We started from skip-gram loss and finish with *t-SNE* like loss

$$L = -\sum_t \sum_{-m\leq j\leq m, j\neq 0} \log\left(p(\mathbf{w}_{t+j}|\mathbf{w}_t)\right) = -\sum_i^V O_i \mathbf{H}\left(\tilde{O}_i, p_i\right)$$

**sum over corpus (scan)**

sum over vocabulary, weighted cross entropy error

**GloVe** - can we use another metric instead of **H**?

**Of course, why not!!!?**

They defined another loss, by replacing cross entropy with other measure of difference between true and learned distributions

The simplest one is least square loss:
where $\mathbf{Q}_{ij}$ and $\mathbf{P}_{ij}$ are unnormalized probabilities

$$L = \sum_{i,j}^V O_i \left(O_{ij} - P_{ij}\right)^2 \quad P_{ij} = \exp\left(\mathbf{w}_i^T \mathbf{w}'_j\right)$$

$$p_{c_i t} = \frac{\exp\left(s_{c_i t}\right)}{\sum_i \exp\left(s_{it}\right)}$$

*softmax is normalized*

$\mathbf{Q}_{ij}$ and $\mathbf{P}_{ij}$ can be very large, hence it is better to compare **log** of them

$$L' = \sum_{i,j}^V f(O_{ij})\left(\log O_{ij} - \log P_{ij}\right)^2 = \sum_{i,j}^V f(O_{ij})\left(\log O_{ij} - \mathbf{w}_i^T \mathbf{w}'_j\right)^2$$

*f is some function, hyperparameter of problem*

# GloVe - relation to word2vec

http://nlp.stanford.edu/pubs/glove.pdf

We started from skip-gram loss and finish with *t-SNE* like loss

$$L = -\sum_t \sum_{-m\leq j\leq m, j\neq 0} \log\left(p(\mathbf{w}_{t+j}|\mathbf{w}_t)\right)$$

**sum over corpus (scan)**

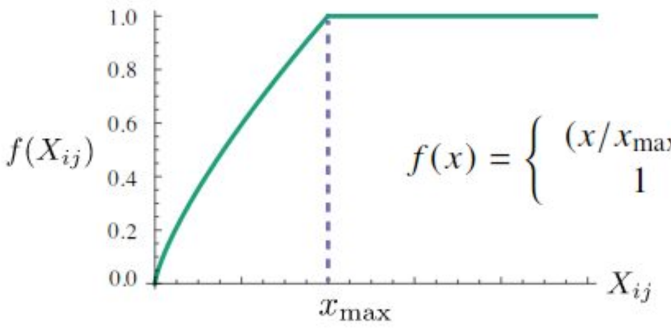
$$= -\sum_i^V O_i \mathbf{H}\left(\tilde{O}_i, p_i\right)$$

sum over vocabulary, weighted cross entropy error

**GloVe** - loss can be obtained from **Skip-Gram** model after few assumptions and phenomenological modifications

$$L_{\text{GloVe}} = \sum_{i,j}^V \boxed{f(O_{ij})} \left(\log O_{ij} - \mathbf{w}_i^T \mathbf{w}_j' - \mathbf{b}_i - \mathbf{b}_j\right)^2$$

**bias vectors**




$$f(x) = \begin{cases} (x/x_{\max})^\alpha & \text{if } x < x_{\max} \\ 1 & \text{otherwise} \end{cases}.$$

Figure 1: Weighting function $f$ with $\alpha = 3/4$.

GloVe - is optimized with SGD

# Wait! Does it look similar to you?

$$L_{\text{GloVe}} = \sum_{i,j}^{V} f(O_{ij}) \left(\log O_{ij} - \mathbf{w}_i^T \mathbf{w}'_j - \mathbf{b}_i - \mathbf{b}_j\right)^2$$

# Collaborative filtering - Implicit feedback

**Collaborative Filtering for Implicit Feedback Datasets**

Yifan Hu
AT&T Labs – Research
Florham Park, NJ 07932

Yehuda Koren*
Yahoo! Research
Haifa 31905, Israel

Chris Volinsky
AT&T Labs – Research
Florham Park, NJ 07932

<- Streszczenie papieru

**Definiujemy** zmienną binarną, (preferencja - p)

$$p_{ui} = \begin{cases} 1 & r_{ui} > 0 \\ 0 & r_{ui} = 0 \end{cases}$$

$r_{ui}$ = np. liczba kliknięcia na produkt i, przez uzytkownika u

oraz poziom pewności w $p_{ui}$

$$c_{ui} = 1 + \alpha r_{ui}$$

Pewność rośnie jak zwiększa się liczba kliknięć

$$L_{\mathrm{GloVe}} = \sum_{i,j}^{V} f(O_{ij}) \left(\log O_{ij} - \mathbf{w}_i^T \mathbf{w}'_j - \mathbf{b}_i - \mathbf{b}_j\right)^2$$

$$c_{ui} = 1 + \alpha \log(1 + r_{ui}/\epsilon)$$

W przypadku dużych różnic w wartościach można stosować ten wzór

Defniujemy stratę do minumalizacji:

$$\min_{x_\star, y_\star} \sum_{u,i} c_{ui}(p_{ui} - x_u^T y_i)^2 + \lambda \left( \sum_u \|x_u\|^2 + \sum_i \|y_i\|^2 \right)$$

<u>GloVe solves same problem as Collaborative filtering with implicit feedback!</u>

poziom pewności stanowi wagę z jaką preferencje danego użytkownika są określane.

**<u>Słowniczek</u>**
**f** - latent factor (10-200)
**gamma** - regularization (0.1-10)
**alfa** - confidence increment (~40)

Yesss! We could use **ALS** solver for finding low dimensional word representations!

Woooow! Finding word representation is actually solving recommendation problem!

# Skip-Gram - relations to matrix factorization

## Neural Word Embedding as Implicit Matrix Factorization


**Omer Levy**
Department of Computer Science
Bar-Ilan University
omerlevy@gmail.com

**Yoav Goldberg**
Department of Computer Science
Bar-Ilan University
yoav.goldberg@gmail.com



### Abstract

We analyze skip-gram with negative-sampling (SGNS), a word embedding method introduced by Mikolov et al., and show that it is implicitly factorizing a word-context matrix, whose cells are the pointwise mutual information (PMI) of the respective word and context pairs, shifted by a global constant. We find that another embedding method, NCE, is implicitly factorizing a similar matrix, where each cell is the (shifted) log conditional probability of a word given its context. We show that using a sparse *Shifted Positive PMI* word-context matrix to represent words improves results on two word similarity tasks and one of two analogy tasks. When dense low-dimensional vectors are preferred, exact factorization with SVD can achieve solutions that are at least as good as SGNS's solutions for word similarity tasks. On analogy questions SGNS remains superior to SVD. We conjecture that this stems from the weighted nature of SGNS's factorization.


- They started with Skip-Gram and Negative sampling method
- And shown that this method is equivalent to factorization of PMI matrix.

Finally, we can describe the matrix $M$ that SGNS is factorizing:

$$M_{ij}^{\text{SGNS}} = W_i \cdot C_j = \vec{w}_i \cdot \vec{c}_j = PMI(w_i, c_j) - \log k$$

where:

Substituting $y$ with $e^x$ and $x$ with $\vec{w} \cdot \vec{c}$ reveals:

$$\vec{w} \cdot \vec{c} = \log \left( \frac{\#(w,c) \cdot |D|}{\#(w) \cdot \#(c)} \cdot \frac{1}{k} \right) = \log \left( \frac{\#(w,c) \cdot |D|}{\#(w) \cdot \#(c)} \right) - \log k \qquad (6)$$

Interestingly, the expression $\log \left( \frac{\#(w,c) \cdot |D|}{\#(w) \cdot \#(c)} \right)$ is the well-known pointwise mutual information (PMI) of $(w, c)$, which we discuss in depth below.

**#(w,c)** - is co-occurrence matrix

$$\text{pmi}(x;y) \equiv \log \frac{p(x,y)}{p(x)p(y)} = \log \frac{p(x|y)}{p(x)} = \log \frac{p(y|x)}{p(y)}.$$

# Epilog GloVe - the fun part

GloVe loss

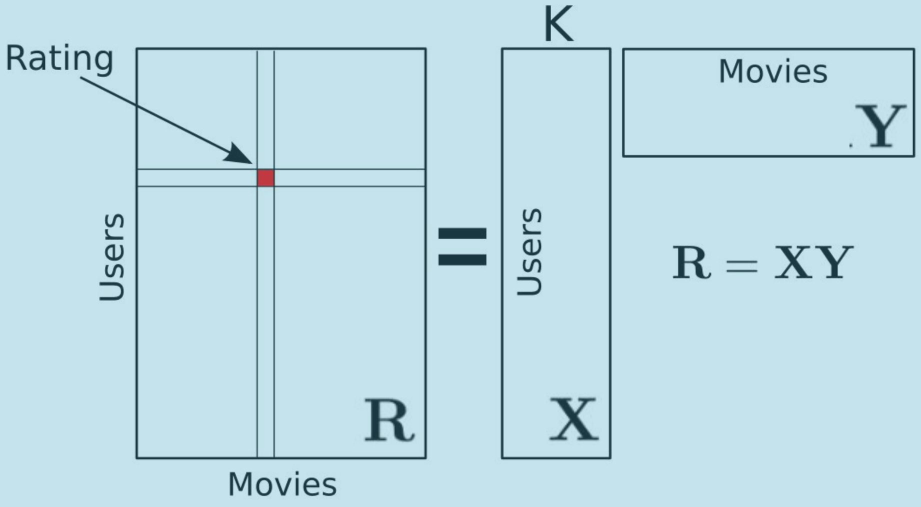
$$L_{\text{GloVe}} = \sum_{i,j}^{V} f(O_{ij}) \left(\log O_{ij} - \mathbf{w}_i^T \mathbf{w}'_j - \mathbf{b}_i - \mathbf{b}_j\right)^2$$

Let’s neglect biases and weight function and regularization term

$$L_{\text{simple}} = \sum_{i,j}^{V} \left(\log O_{ij} - \mathbf{w}_i^T \mathbf{w}'_j\right)^2$$

This can be efficiently minimized with ALS algorithm, used for explicit feedback recommendations

**Note: R**=log(**Q**), **X**=**W** and **Y**=**W’**




$$\mathbf{R} = \mathbf{XY}$$

$$\mathbf{Y} = \left(\mathbf{X}\mathbf{X}^T + \lambda \mathbf{1}\right)^{-1} \mathbf{X}^T \mathbf{R}$$

$$\mathbf{X} = \mathbf{R}\mathbf{Y}^T \left(\mathbf{Y}\mathbf{Y}^T + \lambda \mathbf{1}\right)^{-1}$$

```
In [14]: lambda_ = 0.1
         n_factors = 100
         m, n = Q.shape
         n_iterations = 20

In [15]: X = 5 * np.random.rand(m, n_factors)
         Y = 5 * np.random.rand(n_factors, n)

In [16]: def get_error(Q, X, Y, W):
             return np.sum((W * (Q - np.dot(X, Y)))**2)

errors = []
for ii in range(n_iterations):
    X = np.linalg.solve(np.dot(Y, Y.T) + lambda_ * np.eye(n_factors),
                        np.dot(Y, Q.T)).T
    Y = np.linalg.solve(np.dot(X.T, X) + lambda_ * np.eye(n_factors),
                        np.dot(X.T, Q))
    if ii % 100 == 0:
        print('{}th iteration is completed'.format(ii))
    errors.append(get_error(Q, X, Y, W))
Q_hat = np.dot(X, Y)
print('Error of rated movies: {}'.format(get_error(Q, X, Y, W)))
```

From presentation about recommendations

# Epilog GloVe - the fun part

I’ve solved GloVe loss with ALS

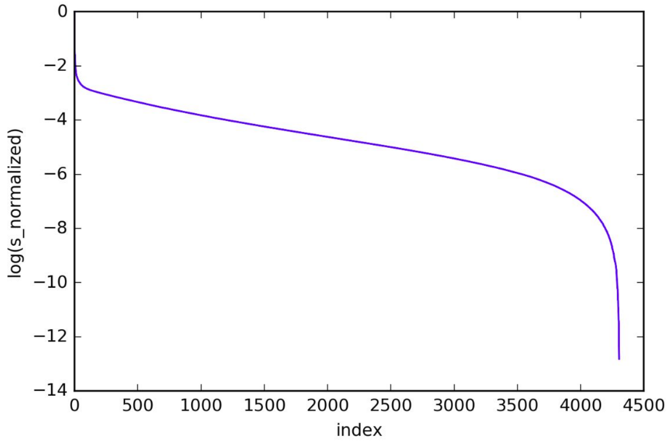
$$L_{\text{simple}} = \sum_{i,j}^{V} \left(\log O_{ij} - \mathbf{w}_i^T \mathbf{w}'_j\right)^2$$

Simple ALS solver in Julia

```
function als(R, lDim, maxIter = 100 , verbose = true)
  nwords = size(R)[1]
  @time X = rand(nwords, lDim)
  @time Y = rand(lDim, nwords)
  i = 0
  eps = 1e-5
  deltaA = 1e6
  deltaB = 1e6
  while i <= maxIter
    i += 1
    deltaA = deltaB
    tmpY = (Y * Y')
    X = R * (Y' * inv(tmpY))
    tmpX = (X' * X)
    Y = transpose(R * (X * inv(tmpX)))
    deltaB = sum(tmpY + tmpX)
    deltaEps = abs((deltaB - deltaA)/deltaB)
    if(deltaEps < eps)
       println("Stopped at intertion ", i, "  eps=", deltaEps)
       break
     end
  end
  return X, Y
end
```

**What I did:**

1. I’ve downloaded all papers of my supervisor from arxiv
2. Extracted all tex files and merged into one file.
3. Then normalized the text: removed most bad words from text.
4. Computed co-occurrence matrix O with window size 10
5. Resulting vocabulary was of size ~4300 words
6. Then created word embeddings using:
   a. Implemented ALS solver (explicit feedback)
   b. Implicit feedback solver (python package)
   c. NMF of co-occurrence matrix (python)
   d. SVD factorization of co-occurrence matrix (python)
   e. word2vec (julia package)




*Normalized singular values of co-occurrence matrix obtained with SVD.*

*I’ve chose latent space size to be **200***

# Epilog GloVe - the fun part

$$L_{\text{simple}} = \sum_{i,j}^{V} \left(\log O_{ij} - \mathbf{w}_i^T \mathbf{w}'_j\right)^2$$

Word: kolasinski

| Row | model_als_X | model_als_X_log | model_ials | model_w2v | nmf_vec | svd_U_vec |
|---|---|---|---|---|---|---|
| 1 | "kolasinski" | "kolasinski" | "kolasinski" | "kolasinski" | "kolasinski" | "kolasinski" |
| 2 | "nowak" | "lent" | "heun" | "radial" | "brun" | "authork" |
| 3 | "mrenca" | "mrenca" | "miseikis" | "detected" | "piazza" | "lent" |
| 4 | "osika" | "piazza" | "pisa" | "beveren" | "beltram" | "roddaro" |
| 5 | "wach" | "bibitemlent" | "mrenca" | "discretized" | "strambini" | "beltram" |
| 6 | "bibitemchwiej" | "kirkner" | "zwierzycki" | "strengths" | "roddaro" | "piazza" |
| 7 | "bednarek" | "beltram" | "bagwell" | "spinconserving" | "heun" | "bibitemlent" |
| 8 | "bibitemszafran" | "authork" | "zebrowski" | "particle" | "poniedzialek" | "mrenca" |
| 9 | "poniedzia" | "roddaro" | "ferry" | "participate" | "silvestro" | "khomyakov" |
| 10 | "bibitemnowak" | "khomyakov" | "nanoscale" | "close" | "pisa" | "kirkner" |

Word: energy

| Row | model_als_X | model_als_X_log | model_ials | model_w2v | nmf_vec | svd_U_vec |
|---|---|---|---|---|---|---|
| 1 | "energy" | "energy" | "energy" | "energy" | "energy" | "energy" |
| 2 | "appear" | "levels" | "potential" | "vppsigma" | "levels" | "levels" |
| 3 | "corresponding" | "spectrum" | "magnetic" | "refosmy" | "spectrum" | "spectrum" |
| 4 | "lowestenergy" | "corresponding" | "field" | "isinfracphi" | "metal" | "electron" |
| 5 | "nearly" | "correspond" | "electron" | "ground" | "kinetic" | "wunsch" |
| 6 | "bright" | "corresponds" | "spin" | "forces" | "plate" | "fermi" |
| 7 | "anticrossing" | "lines" | "system" | "differences" | "transmitted" | "bibitemasm" |
| 8 | "near" | "localized" | "function" | "displayed" | "inducton" | "pereira" |
| 9 | "change" | "energies" | "electrons" | "klein" | "refu" | "calculation" |
| 10 | "appears" | "field" | "wave" | "simulations" | "partial" | "santos" |

**word2vec** ???
very poor results…
in both cases:
probably word2vec
needs more data

# Conclusions

## Count based vs direct prediction

| Count based | Direct prediction |
|---|---|
| LSA, HAL (Lund & Burgess), COALS (Rohde et al), Hellinger-PCA (Lebret & Collobert) | NNLM, HLBL, RNN, Skip-gram/CBOW, (Bengio et al; Collobert & Weston; Huang et al; Mnih & Hinton; Mikolov et al; Mnih & Kavukcuoglu) |
| • Fast training<br>• Efficient usage of statistics<br>• Primarily used to capture word similarity<br>• Disproportionate importance given to large counts | • Scales with corpus size<br>• Inefficient usage of statistics<br>• Generate improved performance on other tasks<br>• Can capture complex patterns beyond word similarity |

**GloVe - is between**

# Conclusions #2

word2vec is not deep learning!

# References


- http://nlp.stanford.edu/pubs/glove.pdf - GloVe paper
- http://www-personal.umich.edu/~ronxin/pdf/w2vexp.pdf - nice tutorial about w2v, well explained with derivations
- https://www.tensorflow.org/tutorials/word2vec/ - tensor flow tutorial about w2v, contains some basic definitions
- https://cs224d.stanford.edu/syllabus.html - Stanford NLP course, contains a lot more information and details
- https://arxiv.org/pdf/1301.3781.pdf - comparision of several techniques like CBOW, Skip-Gram, Nerual Network based…
- http://papers.nips.cc/paper/5021-distributed-representations-of-words-and-phrases-and-their-compositionality.pdf - original Skip-Gram paper
- https://cs224d.stanford.edu/lectures/CS224d-Lecture2.pdf - a lecture (same Stanford course) about w2v and GloVe
- https://radimrehurek.com/gensim/index.html - probably most popular topic modeling library for python
- https://arxiv.org/pdf/1402.3722v1.pdf - Deriving Mikolov et al.'s Negative-Sampling Word-Embedding Method
- https://papers.nips.cc/paper/5477-neural-word-embedding-as-implicit-matrix-factorization.pdf - Paper about relation of Skip-Gram model to Implicit matrix factorization

FORNAX

WWW.FORNAX.CO